SPECIAL

トランジスタ技術

SPECIAL No.49

特集　徹底解説　Z80マイコンのすべて

Z80CPUの概要から周辺LSIの活用法，ICEによるデバッグまで

トランジスタ技術 SPECIAL No.49

CONTENTS

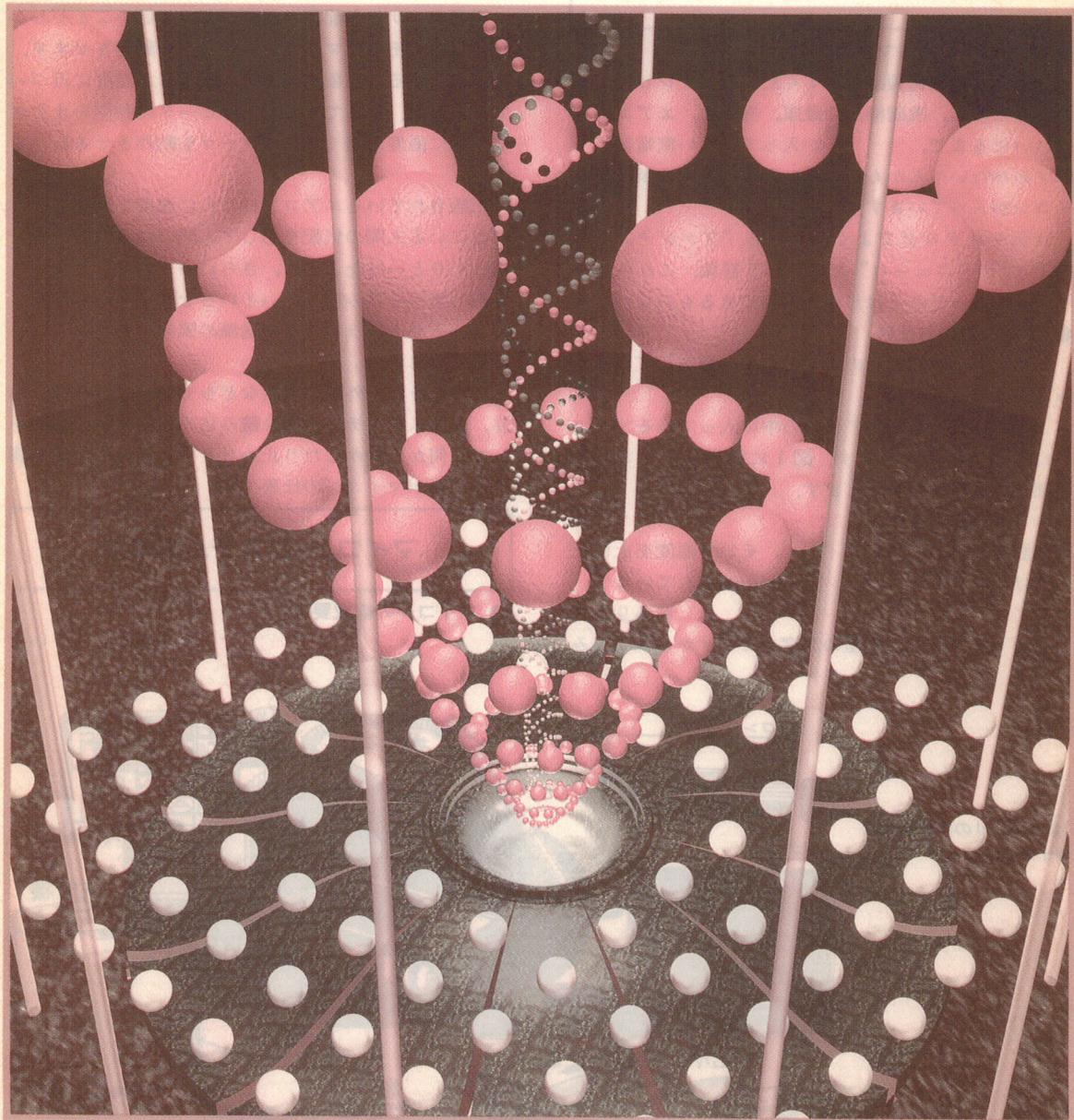
特集　徹底解説　Z80マイコンのすべて

Z80CPUの概要から周辺LSIの活用法，ICEによるデバッグまで

表紙デザイン/簑原圭介　本文イラスト/神崎真理子

特集　徹底解説 Z80 マイコンのすべて

Z80 CPUの概要から周辺LSIの活用法，ICEによるデバッグまで

本誌はＺ80 マイコンの概要からメモリやI/O の接続方法，PIO や CTC，SIO の周辺LSI の使い方，そして周辺 LSI を組み込んだものやＺ80 互換高速 CPU について，その活用方法を徹底解説します．さらにＺ80 を使ったマイコン・システムの開発の流れ，アセンブラの使い方から ICE を使ったデバッグの方法も解説します．

第0章

私たちの生活を支える縁の下の力持ち

マイコンとはなんだろう

野口智樹

家電製品の中に潜むマイコン

● どこにでもあるマイコン

身の回りの家電製品をみてください．テレビやビデオはもちろん，洗濯機，冷蔵庫，掃除機，エアコンなどにいたるまで，マイコンが入っていない家電製品を探すのがたいへんかもしれないほど，多くの製品にマイコンが入っています．

宣伝やカタログでよくみかける“ファジー”だの“AI”だの“ニューロ”だのという言葉，これらはマイコンがあってこそはじめて実現できるものなのです．

● マイコンのお仕事

マイコンは家電製品の中でどんな仕事をしているかみてみましょう．

ここでは例としてエアコンの中に入っているマイコンについて考えてみます．図1に簡単なエアコンの内部ブロックの例を示します．

エアコンには部屋の温度を測定する温度センサ，人間が設定する温度設定スイッチ，そして温度を上げたり下げたりするヒータやクーラが内蔵されています．そしてこれらをコントロールするマイコンがあります．

まずマイコンは，部屋の温度をセンサから取り込みます．そして温度設定スイッチの指示値を読み込んで，測定した温度と比較します．

温度センサから部屋の温度がだいぶ低いことがわかりました．そこでマイコンはヒータを動作させます．しばらくすると部屋が暖まり，指定した温度に近づいてきました．そこでマイコンはヒータの動作を止めます．逆に温度が高くなればクーラを動作させるわけです．

市販されているエアコンはさらに湿度を測定したり，部屋にいる人間の気配を感じて人間のいる方向へ風を送ったりといったように，さらに多くのセンサをつけ外部の情報を読み取り，いろいろなデータから判断してヒータやクーラの温度や送風の風向きをコントロールしています．

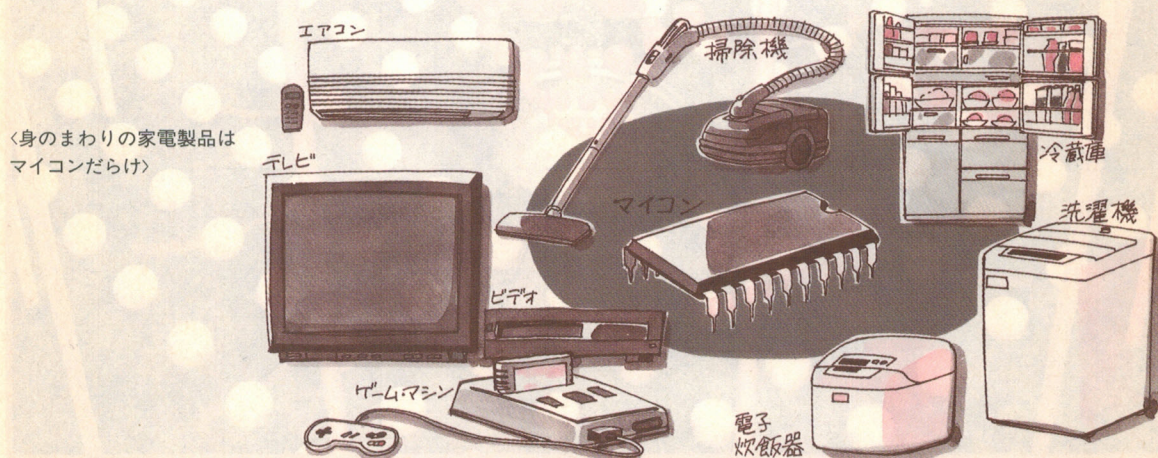
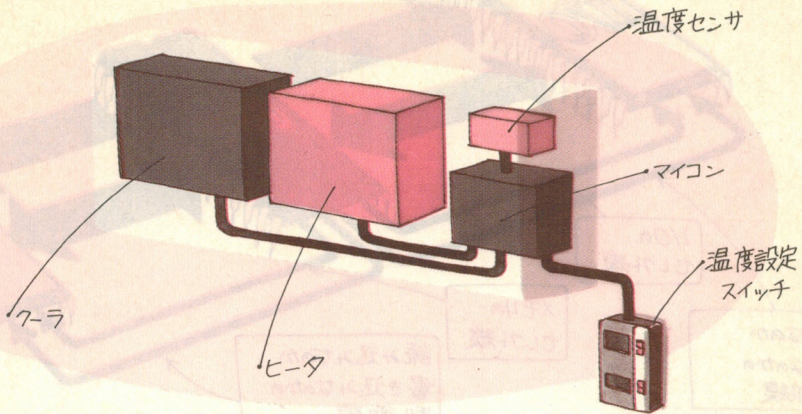
このように，外部の温度などをセンサで読み取ったり，設定スイッチの状態を読み込み，何らかの判断をして，何かを動かしたり止めたりするという作業を繰り返すのがマイコンの仕事です．

マイコン・システムの構成

● マイコンの構成要素

マイコンは三つの基本的な要素から成り立っていま

エアコン
掃除機
冷蔵庫
テレビ
マイコン
洗濯機
ビデオ
ゲーム・マシン
電子炊飯器

〈身のまわりの家電製品はマイコンだらけ〉

〈図 1〉エアコンの内部ブロック図

す．まずいちばん重要なのが，計算をしたり判断をしたりするCPU(中央演算装置)と呼ばれる部分です．そしてそのCPUを動作させるためのプログラムを記憶したり，入力されたデータを保存したりするためのメモリ，もう一つは外部から信号を読み込んだり，外部機器をコントロールするための入出力(I/O)部分です(図2)．

これらの三つのうちどれか一つが欠けても，マイコン・システムは構成できません．また三つの要素があるからといって，かならず3個の部品から成るというわけでもありません．ワンチップ・マイコンとよばれるものは，これら三つの要素が一つのICの中に組み込まれています．

● アドレス/データ・バス，制御線

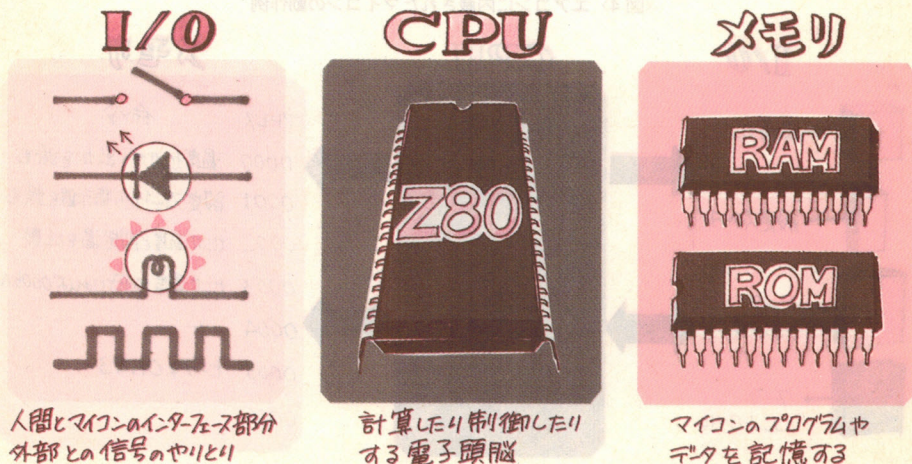
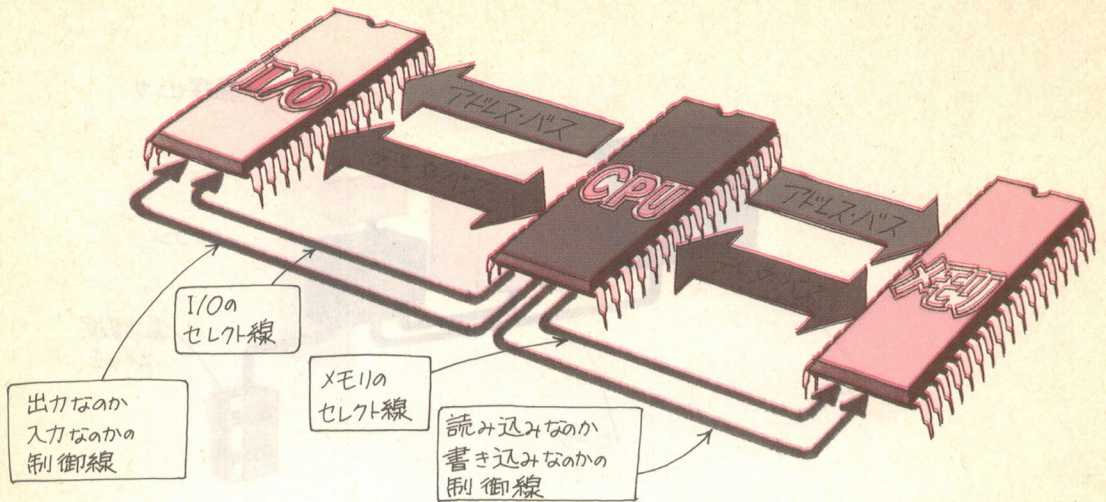
これらCPU，メモリ，I/Oを接続するのがアドレス・バス/データ・バスというデータ線の束です(図3)．

アドレス・バスとは，その名のとおりメモリやI/Oの番地を指定するための線で，データ・バスとは，アドレス・バスで決めたメモリやI/Oの番地にデータを書き込んだり読み込んだりするときの，データの通り道です．

またこれとは別に，メモリなのかI/Oなのかを指定する線，データの読み込みか書き込みかの制御線など，何本かの制御線があります．

● マイコンの基本動作

先ほど説明したエアコンの例に戻しましょう．温度

〈図 2〉マイコンの3要素

〈図3〉 アドレス・バスとデータ・バス

センサからの信号や，人間が設定した温度スイッチの情報の入力，またヒータやクーラのコントロール出力は，マイコンのI/Oに接続されています．

“温度センサの出力を読み込め”や“ヒータをONしろ”というプログラムはメモリに記憶しておきます．

電源ONでエアコンが動き出すと，まずマイコンはアドレス0の命令をメモリから読み込みます．

この命令は“温度センサの出力を読み込め”でした．そこで今度はアドレス・バスに温度センサのI/Oアドレスを出力し，センサからのデータを入力します．

そしてまた次の命令を読み込みます．“温度設定スイッチの指示値を読み込め”でした．今度は温度設定スイッチのI/Oアドレスをアドレス・バスに出力し，設定スイッチの状態を入力します．

またまた次の命令を読み込み，温度センサからのデータと設定スイッチからのデータとどちらが温度が高いかを判断します．

そして温度センサから読み込んだ部屋の温度が低ければアドレス5にジャンプします．

アドレス5の命令は，ヒータをONしろという命令です．アドレス・バスにヒータのI/Oアドレスを出力し，ヒータをONするという意味のFFhというデータを出力します．これでヒータがONされます(図4)．

このようにマイコンは，命令を読み込んで，その命令に従って別のメモリ・アドレスのデータを読み込んだり，I/Oアドレスにアクセスしてデータを入出力します．

そしてふたたび次の命令を読み込んで動作を繰り返します．

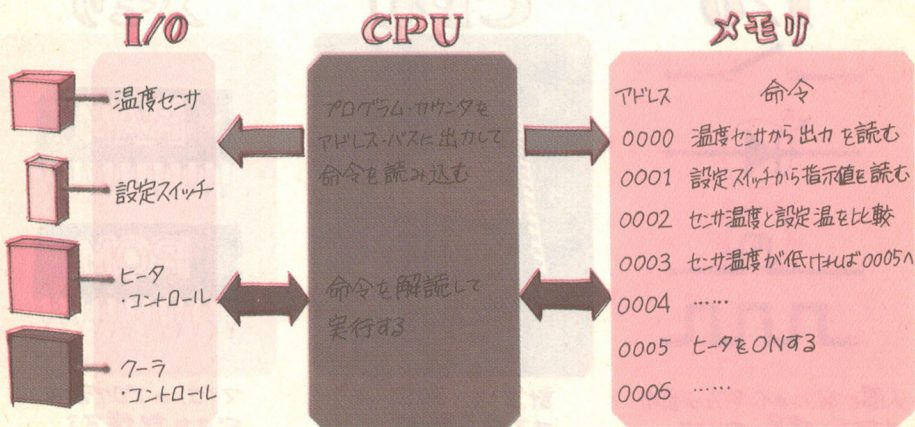
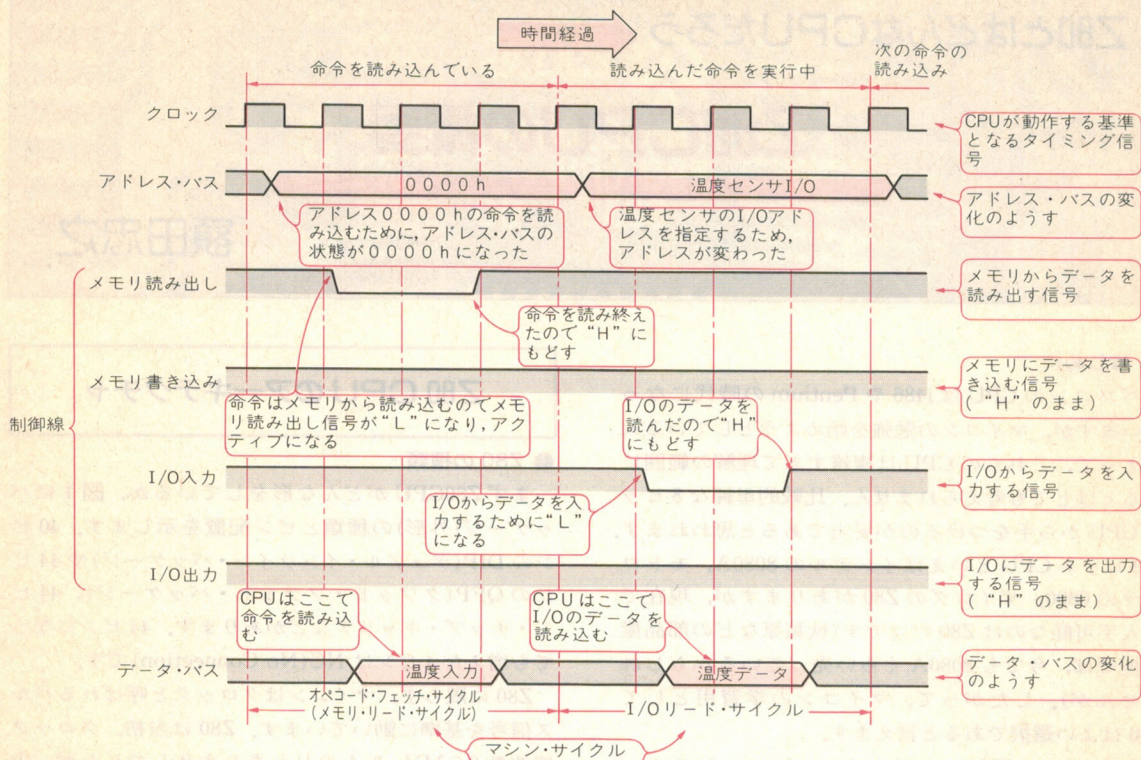
〈図4〉 エアコンに内蔵されたマイコンの動作例

〈図5〉マイコンの動作

● CPU動作のまとめ

図5にCPUの細かな動作の移り変わりを示します．CPUが動作を繰り返すには，クロックと呼ばれる基準信号が必要です．人間でいえば心臓の鼓動のようなものでしょうか．クロック周波数が早ければ，それだけ高速なCPUといえます．

CPUが命令を読み込むには，まず読み込みたい命令のアドレスをアドレス・バスに出力します．命令はメモリから読み出すわけですから，次のクロックのタイミングでメモリ読み出し信号が出力（アクティブ）されます．そして次のクロックのタイミングで，CPUは命令を読み込みます．次のクロックではもう命令を読み込んだ後なので，メモリ読み出し信号を戻します．

これがCPUが命令を読み込むときの動作です．この一連の動作を，命令のことをオペコードと呼ぶことから，オペコード・フェッチ・サイクルと呼びます．またメモリからデータを読み出しているともいえるので，メモリ・リード・サイクルとも呼びます．

さて，次にこの命令の実行です．読み込んだ命令は温度センサのI/Oから温度データを読み込む命令でした．

そこで，まず温度センサのI/Oアドレスをアドレス・バスに出力します．次のクロックではI/O読み出し信号が出力され，次のクロックでCPUはデータを読み込みます．読み込みが終わったので，次のクロックではI/O読み出し信号は元に戻ります．

これで温度センサからのデータの読み込みが終わりました．この動作もI/Oからデータを読み込んでいるので，I/Oリード・サイクルと呼びます．

またこれらのメモリ・リード・サイクルやI/Oリード・サイクルなど，CPUの各行程をマシン・サイクルと呼びます．

ここでアドレス・バスやデータ・バスをみてください．時間経過にしたがって出力されているデータが次々に変化しています．ある瞬間は命令だったり，ある瞬間はI/Oのデータだったりします．これを時分割制御と呼びます．

CPUにはいろいろな種類がありますが，世の中にあるほとんどのCPUは，だいたいこのようなタイミングで動作していると思って間違いありません．

● なぜZ80か

これを否定しまうと，本書の存在意義がなくなってしまうのですが…．

やはりZ80は8ビット・マイコンとして広く使われている，この一言につきると思います．

それでは次章から，Z80CPUの概要や動作について，詳しく解説していきましょう．

（トランジスタ技術1994年6月号に加筆，修正）

第1章

Z80とはどんなCPUだろう

Z80CPUの概要

額田忠之

● はじめに

パソコンのCPUはi486やPentiumの時代になっていますが，マイコンの勉強を始めようとしている人にとって，これらのCPUは複雑すぎて理解の範囲にあるとはとても考えられません．比較的単純な8ビットCPUから手をつけるのが妥当であると思われます．

8ビットCPUといえばインテルの8080A，モトローラの6800，ザイログのZ80がありますが，現在でも入手可能なのはZ80だけです(秋葉原などの部品屋を捜せば，今でも8080Aくらい売っているかもしれませんが)．したがって，マイコンの学習用としてZ80はよい選択であると言えます．

ここでは，Z80とはどんなマイコンなのか，Z80CPUの概要について解説します．

なお余談ですが，日本ではZ80は「ゼット・はちじゅう」または「ゼット・はちまる」と呼ばれていますが，米国人はもちろん「ズィ・エイティ」と呼んでいます．

〈図2〉Z80CPUの信号線

Z80 CPUのアーキテクチャ

● Z80の種類

まずZ80CPUがどんな形をしているか，図1にパッケージ(外形)の種類とピン配置を示します．40ピンのDIP(デュアル・インライン・パッケージ)や44ピンのQFP(クワッド・フラット・パッケージ)，44ピン・チップ・キャリアなどがあります．44ピンになっても増えた4ピンはNC(No Connection)です．

Z80に限らず，マイコンはクロックと呼ばれるパルス信号を基準に動いています．Z80は当初，クロック周波数2.5MHzのものしかありませんでしたが，少しずつ改良されて現在では表1に示すように最高20MHz動作のものまであります．

● Z80の各種信号線

図1に示した外観でわかるように，Z80には40本ものいろいろな信号名のついたピンがあります．それぞれの名称や意味，動作を図2と表2に示し説明します．

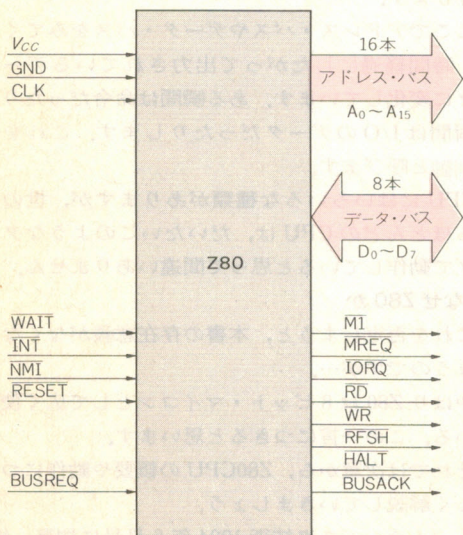
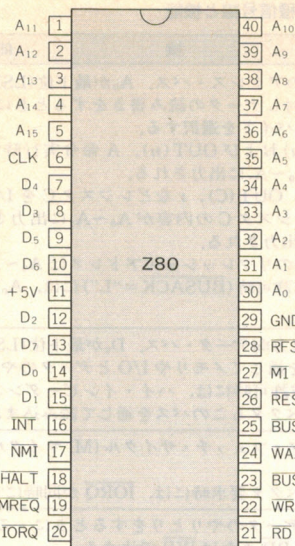
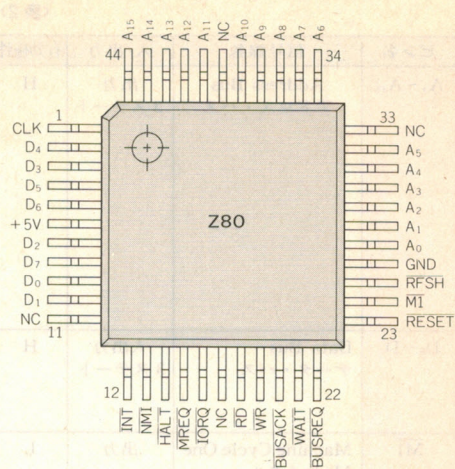
どの信号線もそれぞれ重要な意味があるのですが，ここでは特に重要な信号線について説明します．

▶ アドレス・バス

バスとは信号線の束と考えてください．ということは，アドレス・バスとは番地を示す信号線の束という意味になります．第0章で説明したように，マイコンは命令やデータをメモリから読み込みます．またセンサの信号をI/Oから読み込みます．このとき，どのメモリから読み込むのか，またはどのセンサからデータを読み込むかの番地を指定する必要があります．この指定のためにアドレス・バスが使われます．

Z80にはアドレス・バスが16本あり，それぞれA_0～A_{15}という名前がついていて，2^{16}つまり0000h～0FFFFh(hは16進数を示す添え字)までの64Kバイトのメモリを管理することができるのです．

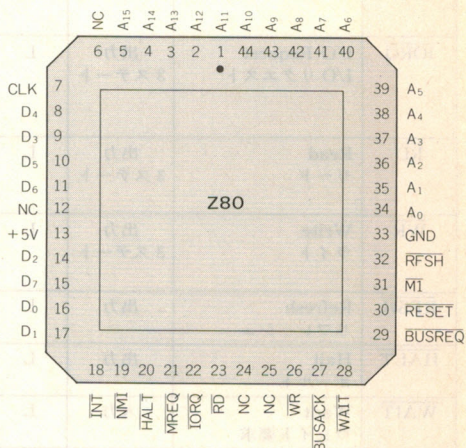
▶ データ・バス

アドレス・バスが番地を示す信号の束なら，データ・バスはデータを示す信号の束となります．メモリ

〈図 1〉
Z80CPU の
パッケージ

(a) 40ピンDIP

(b) 44ピンQFP
(CMOS版のみ)

(c) 44ピン・チップ・キャリア

〈表 1〉 Z80CPU の型名と最高動作速度

NMOS タイプ	最高動作周波数
Z0840004	4 MHz
Z0840006	6.17 MHz
Z0840008	8 MHz

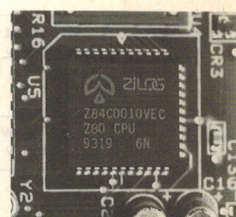
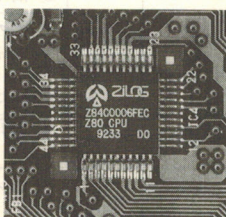
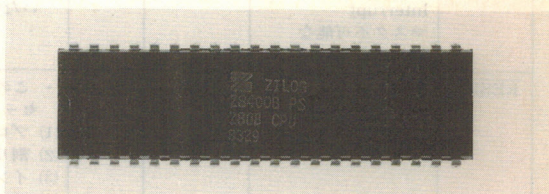
CMOS タイプ	最高動作周波数
Z84C0006	6.17 MHz
Z84C0008	8 MHz
Z84C0010	10 MHz
Z84C0020	20 MHz

から読み込んだ命令やI/Oからのデータは，このデータ・バスを通してCPUに読み込まれます．Z80ではD_0～D_7の8本の線があります．

▶ $\overline{MREQ}$，$\overline{IORQ}$

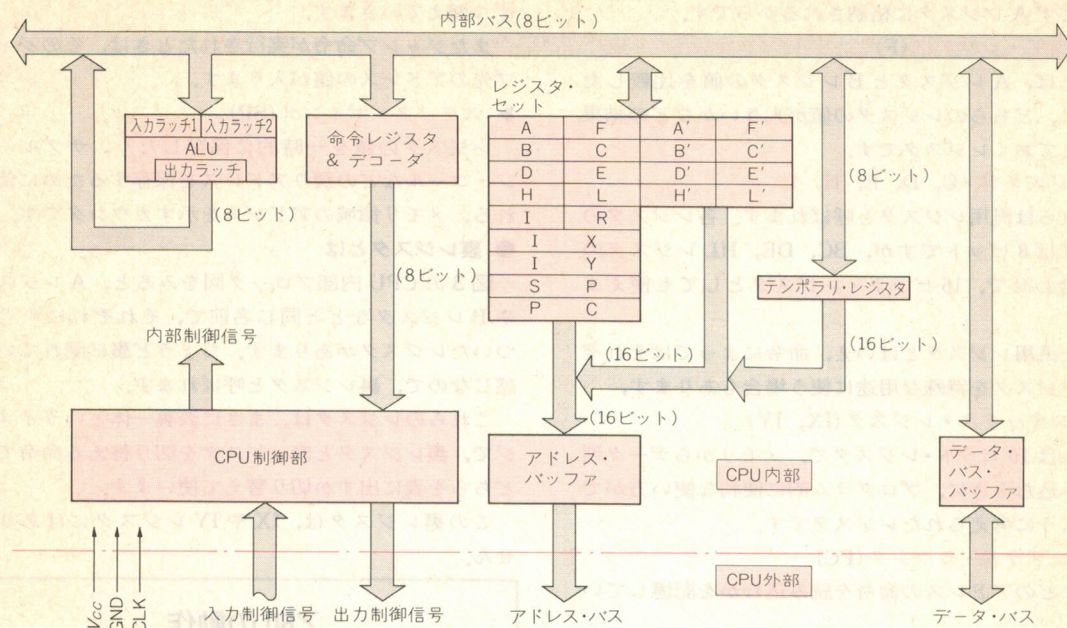
マイコンには命令やデータを読み書きするためのメモリと，センサからの信号を読み込むI/Oがあります．これらの番地の指定にアドレス・バスが使われると説明しました．つまり番地指定にはメモリ，I/Oにかかわらずアドレス・バスが兼用されるわけです．

そこでメモリを読みたいのか，I/Oを読みたいのかという指定を別の信号線で指定する必要があります．そのための信号線がこの$\overline{MREQ}$と$\overline{IORQ}$です．メモリを読み書きしたいときはMREQが，I/Oを読み書きしたいときは$\overline{IORQ}$がLレベルになります．

メモリかI/Oかを区別していますが，ハードウェア的には$\overline{MREQ}$と$\overline{IORQ}$のどちらの信号がアクティブ(有効になる)かという違いだけです．

▶ $\overline{RD}$，$\overline{WR}$

〈写真 1〉 Z80CPU の外観(上：DIP，左下：QFP，右下：チップ・キャリア)

〈表 2〉 Z80CPU の各種信号線と機能

ピン名	信号線名	入/出力	有効極性	機能
A_0～A_{15}	Address Bus アドレス・バス	出力 3 ステート	H	・ 16ビットのアドレス・バス．A_0が最下位(LSB)で，A_{15}が最上位(MSB)． ・ メモリに対しデータの読み書きをするときに，このバスにより 64 K バイト中の1バイトを選択する． ・ IN A，(n)およびOUT(n)，A 命令実行時には，I/O ポート・アドレス，n が A_0～A_7に出力される． ・ IN r, (C)，OUT(C)，r などレジスタ C を I/O アドレスとする入出力命令では，レジスタ C の内容が A_0～A_7に出力され，レジスタ B の内容が A_8～A_{15}に出力される． ・ DRAM 用のリフレッシュ・アドレスは A_0～A_6に出力される． ・ DMA 要求許可時($\overline{BUSACK}$＝“L”)には，A_0～A_{15}はハイ・インピーダンスになる
D_0～D_7	Data Bus データ・バス	入出力 3 ステート	H	・ 8ビットの双方向データ・バス．D_0が最下位(LSB)で，D_7が最上位(MSB)． ・ このバスを通じてメモリや I/O とデータのやりとりをする． ・ DMA 要求許可時には，ハイ・インピーダンスになる． ・ 割り込みベクタもこのバスを通じて読み込まれる．
$\overline{M1}$	Machine Cycle One M1 サイクル	出力	L	・ オペコード・フェッチ・サイクル(M1 サイクル)を実行していることを示す． ・ 割り込みベクタ要求時には，$\overline{IORQ}$ が同時に“L”になる．
$\overline{MREQ}$	Memory Request メモリ・リクエスト	出力 3 ステート	L	・ メモリとデータのやりとりをするときに，この信号は“L”になる．データの方向は $\overline{RD}$ または $\overline{WR}$ で決まる． ・ DRAM 用のリフレッシュ・アドレスを出力するときには $\overline{RFSH}$ 信号とともに“L”になる． ・ DMA 要求許可時には，ハイ・インピーダンスになる．
$\overline{IORQ}$	I/O Request I/O リクエスト	出力 3 ステート	L	・ I/O とデータのやりとりをするときに，この信号は“L”になる．データの方向は $\overline{RD}$ または $\overline{WR}$ で決まる． ・ 割り込みベクタ要求時には，$\overline{M1}$ が同時に“L”になる． ・ DMA 要求許可時には，ハイ・インピー・ダンスになる．
$\overline{RD}$	Read リード	出力 3 ステート	L	・ メモリまたは I/O からデータを読み込むときに“L”になる．メモリと I/O の識別は $\overline{MREQ}$ と $\overline{IORQ}$ により行われる． ・ DMA 要求許可時には，ハイ・インピーダンスになる．
$\overline{WR}$	Write ライト	出力 3 ステート	L	・ メモリまたは I/O に対してデータを書き込むときに“L”になる．メモリと I/O の識別は $\overline{MREQ}$ と $\overline{IORQ}$ により行われる． ・ DMA 要求許可時には，ハイ・インピーダンスになる．
$\overline{RFSH}$	Refresh リフレッシュ	出力	L	・ DRAM 用のリフレッシュ・アドレスが A_0～A_6に出力されていることを示す．
$\overline{HALT}$	Halt ホールト	出力	L	・ HALT 命令が実行したことを示す．この後，割り込みまたは NMI 要求が発生するまで，CPU は NOP 命令の実行を繰り返す．
$\overline{WAIT}$	Wadt ウェイト要求	入力	L	・ CPU がメモリまたは I/O をアクセスするとき，この信号を“L”にすると，CPU はウェイト・サイクルを挿入し，アクセス時間を延長する．
$\overline{INT}$	Interrupt Request 割り込み要求	入力	L	・ この信号が“L”であると，CPU は割り込み受け付け禁止条件が解除されたときに，$\overline{IORQ}$ と $\overline{M1}$ を“L”にすると同時に，割り込みベクタをデータ・バスを通して読み込む．
$\overline{NMI}$	Non Maskable Interrupt マスク不可能な割り込み要求	入力	L	・ この信号の立ち下がりエッジを検出すると，CPU はその時点で実行していた命令の終了時にもっとも優先度の高い割り込みを発生する．
$\overline{RESET}$	Reset リセット	入力	L	・ この信号が“L”であると(最低 3 クロック期間)，CPU はつぎのようにリセットされる． (1) プログラム・カウンタの値はゼロになる． (2) 割り込み受け付けが禁止される． (3) インタラプト・レジスタ(I)がゼロ・クリアされる． (4) リフレッシュ・レジスタ(R)がゼロ・クリアされる． (5) 割り込みモードがゼロ(IM0)になる．
$\overline{BUSREQ}$	Bus Request バス・リクエスト	入力	L	・ この信号が“L”であると，CPU はその時点に実行していたマシン・サイクル終了時に，アドレス・バス，データ・バスならびに制御信号をハイ・インピーダンスにし，バスの制御権を外部回路(例えば DMA コントローラ)に譲る．
$\overline{BUSACK}$	Bus Acknowledge バス・アクノリッジ	出力	L	・ $\overline{BUSREQ}$ 信号が“L”であることを認識し，バスをハイ・インピーダンスにした後，この信号を“L”にしてバスが解放されたことを外部に知らせる．
CLK	Clock クロック	入力	—	・ シングル・フェーズの MOS レベルのクロック． 　　　　　最小　　　　最大 “L”電圧　-0.3 V　　0.45 V “H”電圧　$V_{CC}-0.6$ V　$V_{CC}+0.3$ V

以上の信号線で，メモリかI/Oか，そしてどのアドレスかは指定できます．残るはそのアドレスに対してデータを読み込むのか書き込むのかの信号線が必要です．それがこの$\overline{\mathrm{RD}}$と$\overline{\mathrm{WR}}$です．

▶ $\overline{\mathrm{RESET}}$

これはリセット信号入力です．これをLレベルにするとCPUは初期化状態になり，Hレベルにすると，メモリのアドレス0000hから命令を読み込んで動き出します．

▶ $\overline{\mathrm{M1}}$

これはCPUがメモリから命令を読み込んでいるときにLレベルになる信号線です．

● Z80の内部構成

図3にZ80の内部構成図を示します．CPUをバス幅で分類する呼び方がありますが，Z80は図で示すように，内部バスが8ビットなので，8ビットCPUに分類されています．

ブロック図は複雑そうですが，とりあえず重要な部分を説明します．

▶ ALU(Arithmetic Logical Unit)

日本語訳では中央演算装置で，何のことはない，計算をする部分です．計算といっても，足し算や引き算，論理演算程度しかできません．

実はZ80はかけ算やわり算ができません．これらの計算は，足し算や引き算を組み合わせて計算します．

▶ 命令レジスタ&デコーダ

CPUがメモリから読み込んだ命令が，どんな命令であるかを解読する部分です．ここで命令を解読して，CPU制御部に信号を送ります．

▶ CPU制御部

命令デコーダで解読された命令によって，CPU内部の各部に制御信号を送る部分です．

▶ アドレス・バス・バッファ/データ・バス・バッファ

さきほど説明したアドレス・バスやデータ・バスをコントロールするところです．

アドレス・バスの役割は番地を指定することですから，必ず出力方向になります．

データ・バスはデータを読み込んだり書き込んだりするので，双方向に制御できます．

▶ レジスタ・セット

レジスタとは，メモリから読み出したデータや計算した結果を，CPUがいったん覚えておくところです．

Z80には26個のレジスタがあります．これらのレジスタの中には使い方が完全に決まっているレジスタもあります．また特に決まっていないものを汎用レジスタと呼びます．

▶ テンポラリ・レジスタ

このレジスタがZ80内部に本当に存在するかは不明ですが，これがあると考えたほうが，CPU内部の動作説明が理解しやすくなります．

● Z80の各種レジスタ

それでは，Z80にはどんなレジスタがあるのでしょうか．特に重要なレジスタについて説明します．

▶ アキュムレータ(A)

Z80でいちばん重要なレジスタがこのアキュムレータであるAレジスタです．というのも，Z80の各種

演算は必ずAレジスタに対して行われ，その演算結果も必ずAレジスタに格納されるからです．

▶ フラグ・レジスタ(F)

例えば，AレジスタとBレジスタの値を比較したときに，どちらのレジスタの値が大きいかなどの結果を覚えておくレジスタです．

▶ レジスタB，C，D，E，H，L

これらは汎用レジスタと呼ばれます．各レジスタのサイズは8ビットですが，BC，DE，HLレジスタの組み合わせで，16ビット・レジスタとしても使えます．

また汎用レジスタとはいえ，命令によってはそれぞれのレジスタを特殊な用途に使う場合もあります，

▶ インデックス・レジスタ(IX，IY)

これは16ビット・レジスタで，メモリからデータ列を読み込むときに，プログラム的に便利な使い方ができるように考えられたレジスタです．

▶ プログラム・カウンタ(PC)

次にどのアドレスの命令を読み込むかを記憶しているレジスタです．メモリから命令を読み込むたびに1ずつ増えていきます．

またジャンプ命令が実行されたときは，そのジャンプ先のアドレスの値が入ります．

▶ スタック・ポインタ(SP)

レジスタの値を一時的に保存したり，サブルーチン・コールなどの戻りアドレスを保存するために使われる，メモリ領域のアドレスを示すカウンタです．

● 裏レジスタとは

図3のCPU内部ブロック図をみると，AレジスタやBレジスタなどと同じ名前で，それぞれに“′”のついたレジスタがあります．ちょうど裏に隠れている感じなので，裏レジスタと呼ばれます．

これらのレジスタは，まさに表裏一体というイメージで，裏レジスタと表レジスタを切り替える命令で，どちらを表に出すか切り替えて使います．

この裏レジスタは，IXやIYレジスタにはありません．

Z80の動作

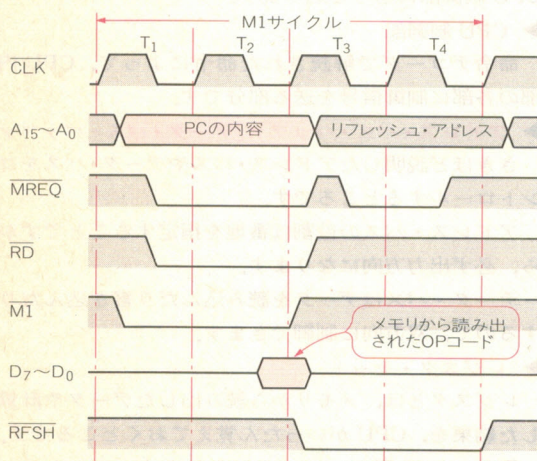
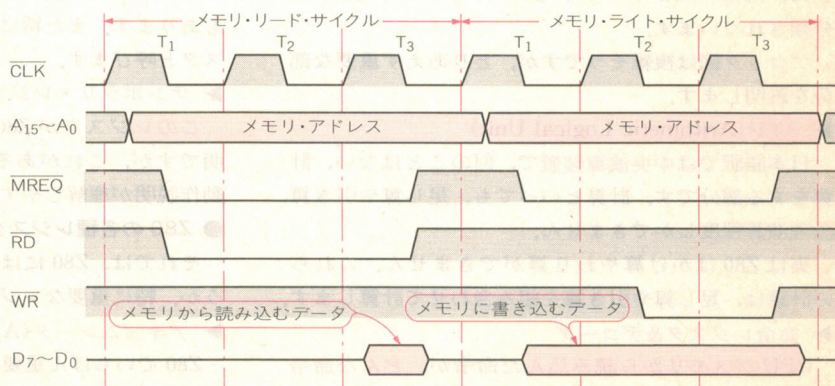
次はZ80の動作についてみてみます．Z80CPUが命令を実行する場合，以下の五つの基本動作(マシン・サイクル)を組み合わせて行います．

(1) オペコード・フェッチ・サイクル(M1サイクル)
(2) メモリ・リード・サイクル
(3) メモリ・ライト・サイクル
(4) I/Oリード・サイクル
(5) I/Oライト・サイクル

では，まずこれらのマシン・サイクルについて説明し，その後で特定の命令がどのように実行されるかを説明します．

● オペコード・フェッチ・サイクル

オペコード・フェッチとは，CPUが命令を読み込むことです．どの命令を実行する場合でも，最初のマシン・サイクルはオペコード・フェッチ・サイクルから始

〈図4〉オペコード・フェッチ・サイクル(M1サイクル)

〈図5〉メモリ・リード/ライト・サイクル

まります．その意味でこのマシン・サイクルはM1サイクルとも呼ばれます．

このサイクルは命令のオペコードをメモリから読み取ります．図4にオペコード・フェッチ・サイクルのタイミングを示します．このマシン・サイクルは，4クロック期間からなり，それぞれのクロック期間にはT_1～T_4という名前がつけられています．

まずT_1のときに，プログラム・カウンタの内容がアドレス・バス(A_0～A_{15})に出力され，ついで$\overline{\text{M1}}$，$\overline{\text{MREQ}}$と$\overline{\text{RD}}$がLレベルになります．

メモリから読み出されたデータ(命令のオペコード)はT_3の立ち上がりでCPU内に取り込まれます．その後，アドレス・バス上にリフレッシュ・アドレスが出力され，$\overline{\text{RFSH}}$および$\overline{\text{MREQ}}$がLレベルになります(DRAMのリフレッシュ・サイクル)．

命令実行時には必ず一つのM1サイクルが実行されますが，このサイクルが2回実行される命令もあります．命令の先頭バイトがDDh，EDh，FDhである命令はオペコードが2バイトになるので，M1サイクルは2回実行されます．

● メモリ・リード/ライト・サイクル

このマシン・サイクルは，命令のオペコードに続くオペランド部分をメモリから読み取るとき，および命令が指示するメモリ・アドレスに対してデータの読み書きをする場合に実行されます．オペランドとは，たとえるなら“Aレジスタと Bレジスタを○○する”という命令のレジスタの指定にあたる部分です．

両サイクルは$\overline{\text{RD}}$が出力されるか，$\overline{\text{WR}}$が出力されるかが異なるだけなので，一緒に説明します．図5にメモリ・リード/ライト・サイクルのタイミングを示します．

メモリ・リード・サイクルでは，T_1のときにメモリ・アドレスがA_0～A_{15}上にのせられた後，$\overline{\text{MREQ}}$および$\overline{\text{RD}}$がLレベルになります．メモリから読み出さ

Z80七不思議　アーキテクチャ編

Z80が誕生して約20年．その間の技術の進歩はめざましいものがあります．このコラムでは，今考えるとなぜZ80ではこうなっているのかといった点について，Q&A方式で考えてみたいと思います．

Q：8080よりZ80はレジスタが強化されているが，AレジスタやHLレジスタをバンク切り替え方式にしたのはなぜか．EXX命令で切り替えないとアクセスできないのでは面倒だし，一概にレジスタが増えたとはいいがたい気がする．

A：内部回路を見ると，それぞれのレジスタを2組もたせ，フリップフロップを命令により切り替えるという文字どおりのバンク切り替えとなっており，もっとも簡単に増設する方法をとったことがうかがえます．

このデバイスの開発当時は，できるだけゲート数を少なくするということが絶対の時代でしたので，裏レジスタをサポートする命令もそのために組み込まれなかったと思われます．

Q：演算命令がアキュムレータに集中していて命令の直交性が悪い．8080からZ80に強化するときに，2バイト命令になったとしても，ADD B，Cなどのようにしていれば，命令も覚えやすくプログラムも作りやすくなったと思う．

A：Z80CPUの開発目標の一つは，8080との命令セットの互換性ということにありました．また，8080系のCPUは，専用レジスタのアーキテクチャをとっており，各レジスタの性格がはっきり決まっています．良かろうが悪かろうが互換性がないことには始まらないため，8080のアーキテクチャをそのまま引きずってしまったのだと思います．

また当時では，8080との互換性を保った状態で汎用レジスタ・アーキテクチャに変えることは，ゲート数の面からも難しかったと考えます．

Q：なぜ制御線が，$\overline{\text{MRD}}$，$\overline{\text{MWR}}$，$\overline{\text{IORD}}$，$\overline{\text{IOWR}}$(メモリ・リード/ライト，I/Oリード/ライト)ではなく，$\overline{\text{MREQ}}$，$\overline{\text{IORQ}}$，$\overline{\text{RD}}$，$\overline{\text{WR}}$なのか．これではメモリをつなぐにも必ずORゲートが必要になってしまう．

A：これは，Z80周辺LSIの制約の関連からかと思います．Z80周辺LSIでは，CPUとのインターフェースに使用する信号線の数を節約するため，各信号に複数の線の組み合わせで各種トランザクションの区別を行うようになっています．$\overline{\text{IOWR}}$や$\overline{\text{IOWR}}$では，I/OリードまたはI/Oライトのときの状態しか示せませんし，両方がイネーブルにはなりえません．これではCPUの状態や割り込みベクタ要求などを周辺LSIが知ることができません．〈信垣育司〉

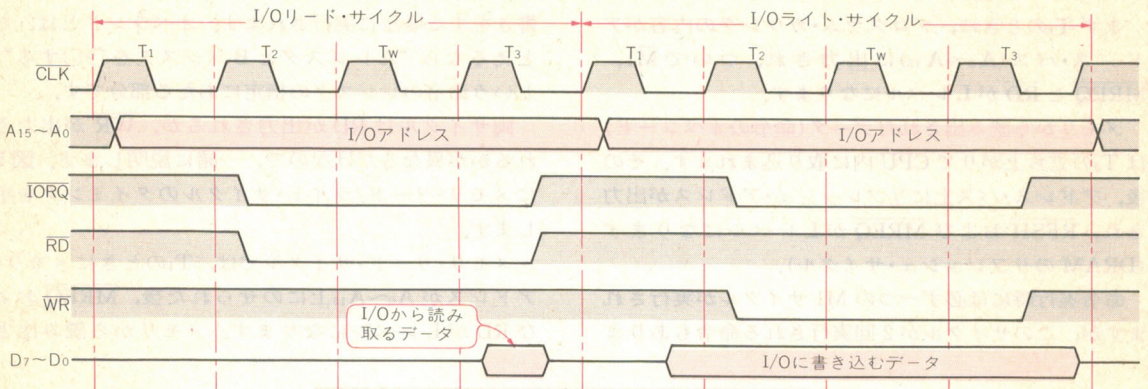
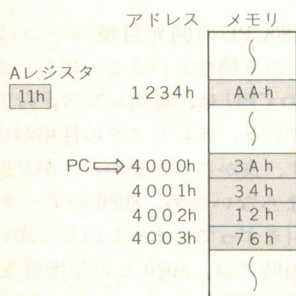
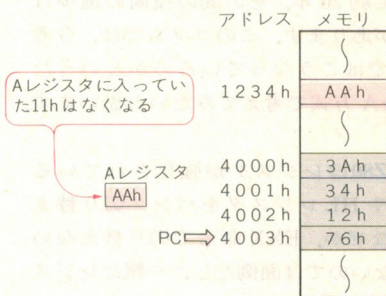
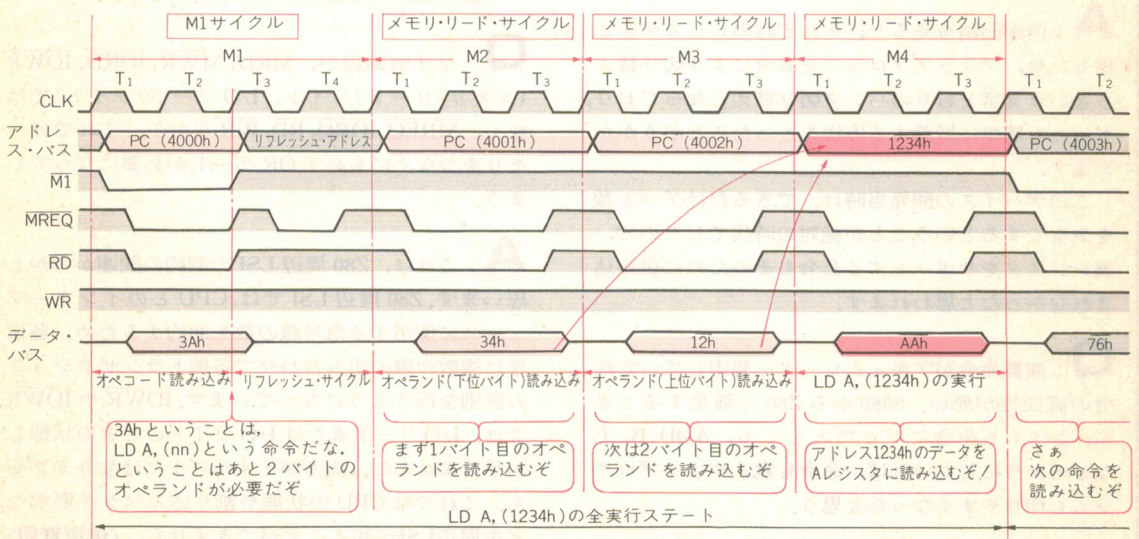
れたデータはT_3の立ち下がりで読み込まれます．

またメモリ・ライト・サイクルでは，T_1のときにメモリ・アドレスがA_0～A_{15}上にのせられ，$\overline{MREQ}$がLレベルになります．また，メモリに書き込まれるべきデータがD_0～D_7上にのせられます．そして，T_2の立ち上がりで$\overline{WR}$がLレベルになります．

● I/Oリード/ライト・サイクル

このマシン・サイクルもリード/ライト・サイクルは似ているので一緒に説明します．

このサイクルはIN命令またはOUT命令の実行時

〈図6〉 I/O リード/ライト・サイクル

〈図7〉 LD A, (nn)命令の実行のようす

(a) 実行前のメモリとレジスタ

(c) 実行後のメモリとレジスタ

(b) バスの動作のようす

に行われます．図6にI/Oリード/ライト・サイクルのタイミングを示します．

I/Oリード・サイクルでは，T_1のときにI/OアドレスがA_0～A_7(正確にはA_0～A_{15})にのせられ，T_2の立ち下がりで$\overline{IORQ}$と$\overline{RD}$がLレベルになります．

I/Oから読み取られるデータはT_3の立ち下がりで取り込まれます．また，I/Oライト・サイクルでもT_1のときにI/OアドレスがA_0～A_7(正確にはA_0～A_{15})にのせられ，I/Oに書き込むデータがD_0～D_7上にのせられます．そして，T_2の立ち下がりで$\overline{IORQ}$と$\overline{WR}$がLレベルになります．

両マシン・サイクルともT_2とT_3の間にT_wというステートがありますが，これはCPUが自動的に挿入するウェイト・ステートで，あまりアクセス・タイムの速くないMOSのI/Oなどに対処するためです．

● マシン・サイクルの実行例

ここでは以上で説明した事がらをさらにわかりやすくするため，実際の命令を実行する場合のマシン・サイクルの動きについて見てみます．とりあげる命令は，

(1) LD　A, (nn)
(2) LD　(nn), A
(3) IN　A, (n)
(4) OUT　(n), A

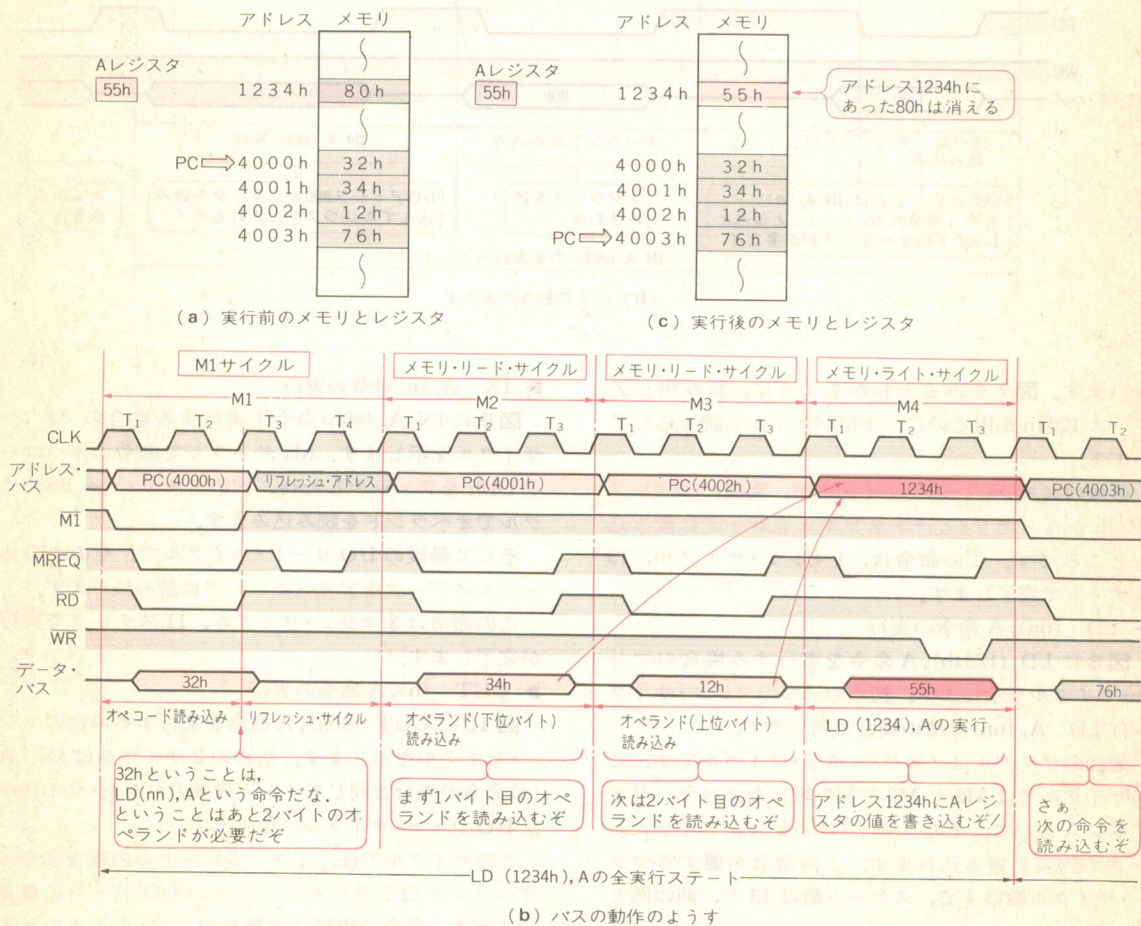
です．

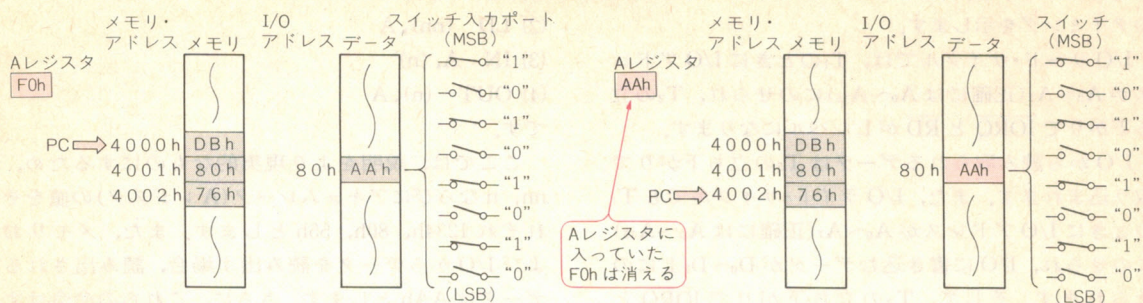
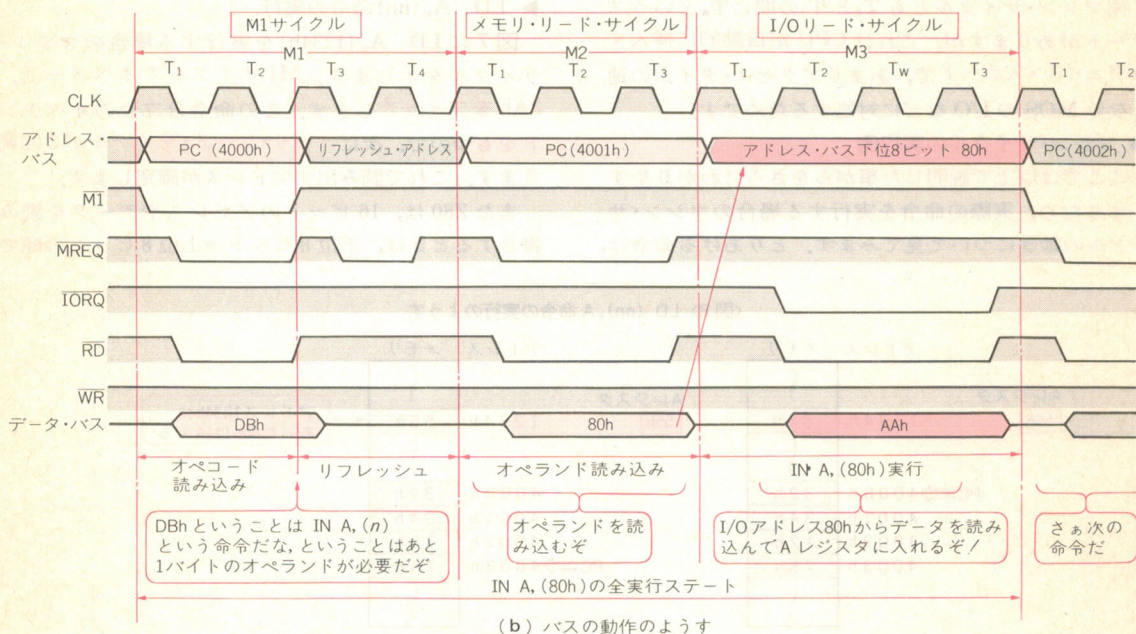
ここでは，説明をより現実的なものにするため，nn，nならびにアキュムレータ(Aレジスタ)の値をそれぞれ1234h，80h，55hとします．また，メモリおよびI/Oからデータを読み出す場合，読み出されるデータはAAhとします．さらに，これらの命令はすべて4000h番地から始まるものとします．

▶ LD　A, (nn)命令の実行

図7にLD　A, (1234h)を実行する場合のマシン・サイクルを示します．M1サイクルでオペコード，3Ahをフェッチします．この命令は二つのオペランドをもつので，次にメモリ・リード・サイクルが2回続きます．これで読み出すアドレスが確定します．

またZ80は，16ビットのアドレスやデータを読み書きするときは，下位8ビット→上位8ビットの順で

〈図8〉 LD (nn), A命令の実行のようす

(b) バスの動作のようす

〈図9〉 IN A, (n)の実行のようす

（a）実行前のメモリとI/Oのようす　（c）実行後のメモリとI/Oのようす

（b）バスの動作のようす

行います．図7をみるとわかるように，読み出しアドレス1234hを得るのに，下位バイトから読み込んでいます．

最後のメモリ・リード・サイクルが，実際にアドレス1234hからメモリ・データをアキュムレータに読み込むところです．この命令は，4マシン・サイクル，13ステートで完了します．

▶ LD　(nn), A 命令の実行

図8に LD (1234h), A 命令を実行する場合のマシン・サイクルを示します．初めの三つのマシン・サイクルは LD　A, (nn)命令の場合と同じです．

最後のサイクルはメモリ・ライト・サイクルです．このサイクルでは M2 と M3 で読み取ったオペランドをアドレスとして，そのメモリ・アドレスにアキュムレータの内容を書き込みます．この命令が要するマシン・サイクル数は4で，ステート数は13と，前の例と同じです．

▶ IN　A, (n)命令の実行

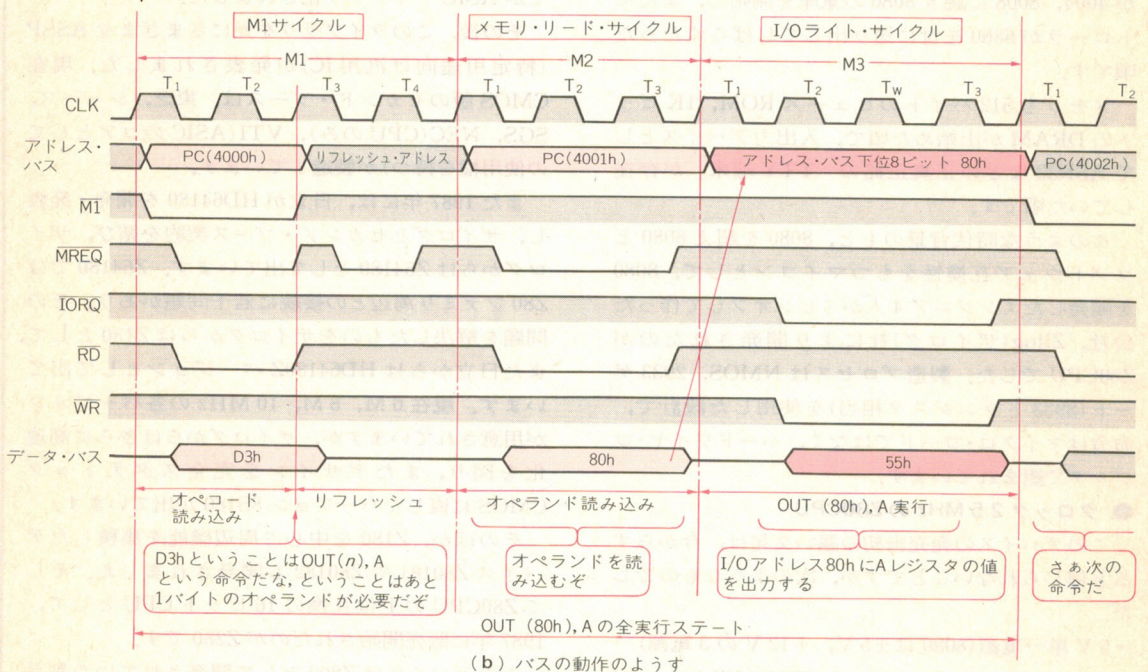
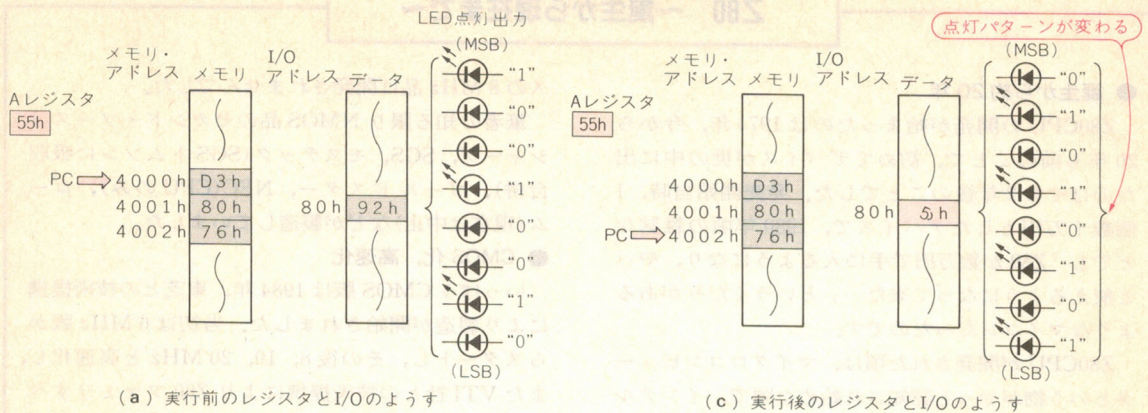
図9に IN A, (80h)命令を実行する場合のマシン・サイクルを示します．M1 サイクルで命令のオペコード DBh をフェッチした後，次のメモリ・リード・サイクルでオペランドを読み込みます．

そして最後のI/Oリード・サイクルで，そのI/Oポートからデータをアキュムレータに読み込みます．

この命令は3マシン・サイクル，11ステートで実行が完了します．

▶ OUT　(n), A 命令の実行

図10に OUT (80h), A 命令を実行する場合のマシン・サイクルを示します．初めの2サイクルは IN　A, (n)命令の場合と同じですが，最後のマシン・サイクルはI/Oライト・サイクルになります．

このサイクルでは，アキュムレータの内容を命令のオペランドで示されるアドレスのI/Oポートに書き込みます．命令の実行に必要なマシン・サイクルとス

テート数はIN命令の場合と同じです．

● **Z80の割り込みシステム**

マイコン・システムにおいてもう一つ重要な概念があります．第6章で詳しく解説しますが，割り込みという概念です．

Z80ではこの割り込みについても，Z80ファミリと呼ばれるZ80用に作られた周辺LSIと組み合わせることで，非常に強力な割り込みシステムを構築できます．

*　　　　*

● **おわりに**

筆者は10年ほど前に著した書籍の冒頭に，「Z80は永遠である」と書きましたが，永遠とはいかなくても，現在までZ80が建在なのを知り，本当に喜んでいます．

4040に始まり，8080A，6800，8085，Z80，8086，80x86と種々のCPUと付き合ってきましたが，Z80は今でも心に残るCPUです．

◆参考文献◆

(1) VOLUME 1 DATEBOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.
(2) Z80 Family User's Manual, ZILOG.

（トランジスタ技術1994年6月号に加筆，修正）

Z80 ～誕生から現在まで～

● 誕生から約20年

Z80CPUの開発が始まったのは1974年，今から20年も前のことで，初めてデバイスが世の中に出たのはその2年後のことでした．発売開始当時，1個数十万円もしたデバイスで，1980年頃の雑誌などでも「Z80が数万円で手に入るようになり，やっと使えるようになって来た…」というくだりがあるようなマイコンだったのです．

Z80CPUが開発された頃は，マイクロコンピュータという物がやっと認知され始めた頃で，インテルが4004，8008に続き8080の量産を開始し，またモトローラが6800を世に送り出してしばらくたった頃です．

メモリも512バイトのヒューズROM，1Kビットの DRAMが出始めた頃で，入出力デバイスとしてASR33などの正真正銘の「TTY端末」が存在していた頃です．

このような時代背景のもと，8080を超え8080とソフトウェア互換性をもつマイコンとして，8080を開発したエンジニア4人がスピンオフして作った会社，Zilog(ザイログ)社により開発されたのがZ80CPUでした．製造プロセスはNMOS，2933ゲート(8933トランジスタ相当)を使用した設計で，命令はマイクロ・コードではなく，ハードワイヤ・ロジックで組まれています．

● クロック2.5 MHzのZ80CPU

このデバイスの発売当初の謳い文句は，今からすると信じられないことですが，次のようなものでした．

- 5V単一電源(8080は±5V，+12Vの3電源)
- シングルフェーズ・クロック(8080は2相クロック)
- 強化された割り込み(モード1，2割り込み)
- 増強されたレジスタ・セット(裏レジスタおよびインデックス・レジスタ)
- 強化された158種の命令(8080は公称78種類)

発売当初，プロセスはNMOS，動作クロックは2.5 MHzでした．その後，クロックも4 MHz(その当時はAバージョンと呼ばれていた)，6 MHz(同Bバージョン)とクロックも早くなっていき，1984年にはHバージョンと呼ばれるクロック8 MHzの製品も発表されました．

いっぽうの周辺デバイスは，1970年代後半にはすべてそろい，CPUクロックの高速化に歩調をあわせ，4 MHz，6 MHzのバージョンが世に送り出されていきました．残念ながらNMOS周辺デバイスの8 MHz品は開発されませんでした．

筆者の知る限りNMOS品のセカンド・ソースは，シャープ，SGS，モステック(SGSトムソンに吸収合併)，ゴールドスター，NEC(CPUのみ)，ローム(現在は中止)などが製造していました．

● CMOS化，高速化

いっぽうCMOS版は1984年，東芝との技術提携により製造が開始されました．当初は6 MHz版からスタートし，その後8，10，20 MHzと高速化し，またVTI社との技術提携によりZ80ファミリすべてがASICライブラリ化されました．

その後，このライブラリを元にさまざまなASSP(特定用途向け汎用IC)が発表されました．現在CMOS版のセカンド・ソースは，東芝，シャープ，SGS，NEC(CPUのみ)，VTI(ASICのコアとしての使用権を持つ)が製造しています．

また1987年には，日立がHD64180を開発・発表し，ザイログとセカンド・ソース契約を結び，ザイログからはZ64180として出ています．Z64180ではZ80ファミリ周辺との接続に若干問題があり，その問題を解決したものをザイログからはZ180として，また日立からはHD64180Zバージョンとして出ています．現在6 M，8 M，10 MHzの各バージョンが用意されていますが，ザイログからはさらに高速化を図り，またデザインを完全スタティックCMOSに直したバージョン8S180が出ています．

そのほか，Z180を中心に周辺機能を集積したデバイスZ80181やZ80182も開発されました．そしてZ80CPUの上位互換の16ビットCPUとして，1987年に販売開始されたのがZ280です．

このデバイスはZ800として開発されていた製品で，製品概要が発表されてからなかなか現物が出てこなかったため，「嘘八百」と陰口を叩かれていた製品でもありました．

そのアーキテクチャから，Z80CPU用に書かれたアプリケーションをそのまま走らせた場合，それほどパフォーマンスが上がらなかったことから，幅広く採用されるまでには至りませんでした．

そして現在，その欠点を克服し16/32ビット・アーキテクチャで新たに設計されたデバイスが1993年に発表されたZ380です．

またR800(アスキー三井セミコンダクター)やKL5C8012(川崎製鉄)などのようなZ80ソフトウェア互換の高速CPUも登場しています．

〈信垣育司〉

第2章

Z80を自由自在に動かすための

Z80のレジスタと命令

野口智樹

Z80のレジスタ

● レジスタとは

レジスタとはお店にあるレジと同じ意味で，値を格納するためのものです．値を格納するだけならメモリ(RAM)と同じですが，メモリに対しての値の格納よりも高速かつ簡単に操作が行えます．

Z80にはいろいろな種類のレジスタが用意されています．第1章でも少し触れましたが，ここでは特にソフトウェア的なところから解説します．図1にZ80のレジスタ・セットを示します．

▶ フラグ・レジスタ(F)

フラグ・レジスタそのものは8ビットのレジスタですが，その内容の1ビットごとがそれぞれのフラグとしての意味をもっています．

フラグとは演算などの実行結果の状態を示すものです．フラグ・レジスタに関する詳細は後述します．

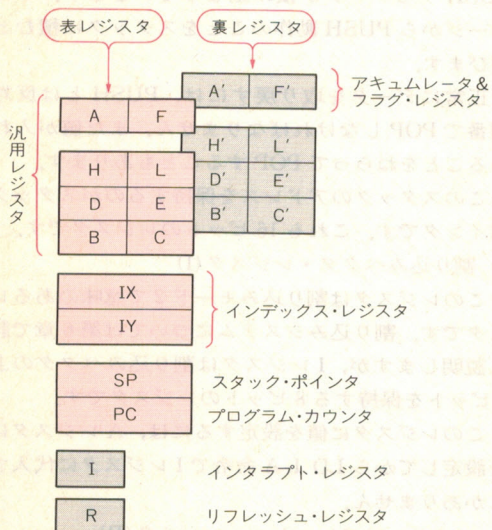
▶ 汎用レジスタ(A, B, C, D, E, H, L)

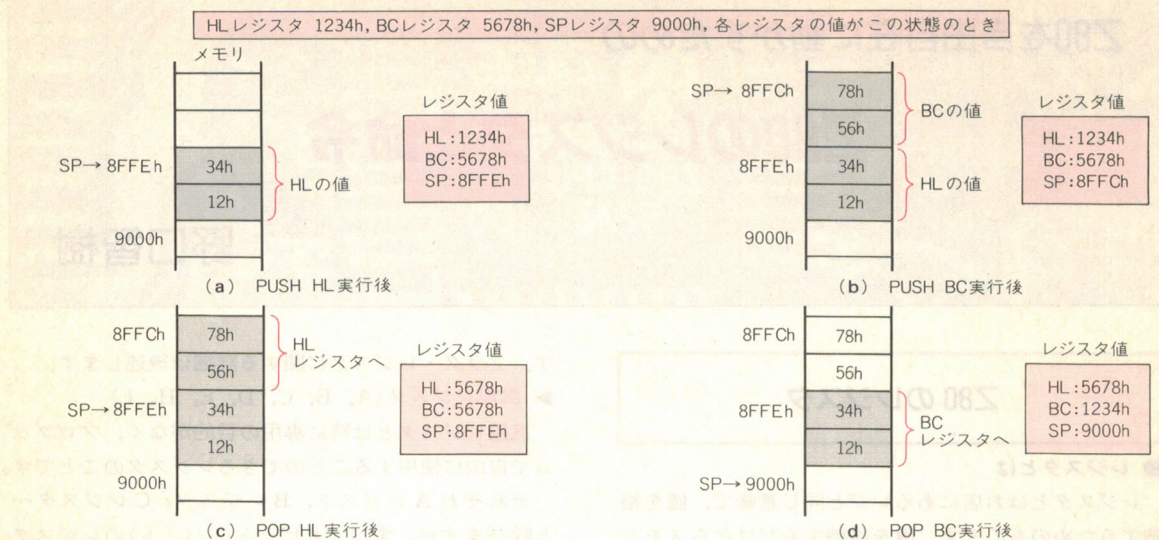
汎用レジスタとは特に専用の目的がなく，プログラムで自由に使用することのできるレジスタのことです．

それぞれAレジスタ，Bレジスタ，Cレジスタ…と呼びますが，すべて8ビット(1バイト)のレジスタです．特にAレジスタ以外のB, C, D, E, H, Lは，BC, DE, HLといった具合に二つを組み合わせて16ビット(2バイト)の値を扱えます．これらをペア・レジスタといいます．

Aレジスタは前述のフラグ・レジスタ(F)とペアになりますが，AFの形では16ビットの値は扱えません(PUSH/POP/EXなどの命令でペアになる)．

また，特に専用の目的がないと説明しましたが，Bレジスタが8ビットのカウンタに，CレジスタがI/Oポートのアドレス指定に，それぞれ使われる専用の命令も一部にはあります．

〈図1〉Z80のレジスタ・セット

〈図2〉EXとEXX命令の動作

〈図3〉 PUSH, POPの動作

さらに汎用レジスタのうち，特にAレジスタはアキュムレータと呼ばれ，ADD/SUB/AND/OR/XORなどの演算結果を直接格納できる唯一のレジスタです．

▶ 裏レジスタ (AF′, BC′, DE′, HL′)

Z80では前述したフラグ・レジスタと汎用レジスタと同様のものをもう1組もっており，これを裏レジスタといいます．裏レジスタだからといって使い方に特に違いはありません．

EX AF, AF′命令でA, Fレジスタを，EXX命令でB, C, D, E, H, Lレジスタの裏と表を切り替えることができます(図2)．

このように切り替えて使用するので，裏レジスタのBレジスタと表のAレジスタをダイレクトに足し算するという意味のADD A, B′などという命令はありません．

このときはEXX命令で，裏レジスタを表にレジスタに切り替えてからADD A, Bを実行します．

▶ インデックス・レジスタ (IX, IY)

インデックス・レジスタにはIXレジスタとIYレジスタの二つがあり，これらは共に16ビットのレジスタです．HLレジスタのように，場合によってH, Lと8ビットずつに使用することは(正式には)できません．

またアドレッシング・モードのところで説明しますが，このレジスタはその名前のとおりインデックスとしての使い方ができるようになっています．

▶ プログラム・カウンタ(PC)

プログラム・カウンタは現在実行中のプログラムのアドレスを示します．

Z80のメモリ空間は64Kバイト(0000～0FFFFh)と16ビットで表現できますから，当然PCは16ビット・レジスタです．

▶ スタック・ポインタ(SP)

スタックとはひとことでいうと先入れ後出しの一時格納庫のことです．

スタック操作命令の代表としてPUSH/POP命令があります．例えばPUSH BCと実行すればBCの内容が一時的にスタックというデータ領域に格納されます．これを再度取り出すにはPOP BCとします．

ここでPUSH　HL, PUSH　BCを実行したあと，POP HL, POP BCを実行すると，BCとHLの内容が入れ替わってしまいます．図3をみてください．

このように，先にPUSHしたデータの上に次にPUSHするデータが積み重なっていきます．このイメージからPUSH動作のことをスタックに積むとも呼びます．

正常にデータを取り戻すには，PUSHとは反対の順番でPOPしなければなりません．また値が入れ替わることをねらってPOPすることもあります．

このスタックのアドレスを保持するのがスタック・ポインタです．これも16ビットのレジスタです．

▶ 割り込みベクタ・レジスタ(I)

このレジスタは割り込みモード2で意味のあるレジスタです．割り込みシステムについては第6章で詳しく説明しますが，Iレジスタは割り込みベクタの上位8ビットを保持する8ビットのレジスタです．

このレジスタに値を設定するには，Aレジスタに値を設定してからLD I, A命令でIレジスタに代入するしかありません．

▶ メモリ・リフレッシュ・レジスタ(R)

本来はDRAMのリフレッシュ制御のためのレジスタですが，現在では実際にそのような用途で使用する

〈表 1〉 キャリ・フラグとゼロ・フラグの動作

実行前の Aレジスタ	命令	実行後の Aレジスタ	実行後のフラグ	
			キャリ・フラグ	ゼロ・フラグ
FFh	SUB 1	FEh	0	0
00h	SUB 1	FFh	1	0
01h	SUB 1	00h	0	1
02h	SUB 1	01h	0	0
FEh	ADD A,1	FFh	0	0
FFh	ADD A,1	00h	1	1
00h	ADD A,1	01h	0	0
01h	ADD A,1	02h	0	0

ことはまずありません．

このレジスタは8ビットですが，このうち最上位ビット(ビット7)の状態は変化しません．下位7ビットが規則的に刻々と変化するレジスタであり，この性質を利用して乱数の生成などで使用することもできます．

Z80のフラグの動作

● フラグの役割

CPUは演算を行った結果，判定を行うために各種のフラグが用意されています．フラグとは“1”か“0”の二つの状態のいずれかを示すものです．

Z80には8ビットのフラグ・レジスタのうち，6ビットが意味のあるフラグとして割り当てられています．

6ビットのフラグのうち，特に重要なのはゼロ・フラグ(Z)，キャリ・フラグ(C)，サイン・フラグ(S)，パリティ/オーバ・フロー・フラグ(P/V)の四つです．

これらのフラグがどのビットであるかということは重要ではありませんが，各フラグの示す意味や演算結果によるフラグの変化については，しっかりと理解しておく必要があります．

● キャリ・フラグ(C)

これは演算の結果，桁あふれが生じたことを示すフラグです．キャリ・フラグは符号なし演算の場合に意味があります．

例えば，8ビットの符号なし数値の最大値は0FFh(255)ですが，加算命令などでこれを越えるような演算の結果が生じた場合にキャリ・フラグが1になります．また，それとは逆に減算命令などで0より小さい結果が生じた場合にも同様にキャリ・フラグが1になります．

キャリ・フラグは，ローテイト・シフト命令などでも使われます．

なお，キャリ・フラグは，SCF命令でセット(1)，CCF命令でリセット(0)と直接コントロールすることもできます．

● ゼロ・フラグ(Z)

これは演算結果がゼロであることを示すフラグです．

〈表 2〉 サイン・フラグの動作

符号なしの値	符号付きの値	サイン・フラグ	2進数表現
0	0	0	00000000
1	1	0	00000001
2	2	0	00000010
∫	∫	∫	∫
127	127	0	01111111
128	−128	1	10000000
129	−127	1	10000001
∫	∫	∫	∫
254	−2	1	11111110
255	−1	1	11111111

例えば，いまAレジスタに1が入っているとしてDEC A(Aを−1する)命令を実行するとAレジスタは0になり，ゼロ・フラグは1になります．

逆に現在Aレジスタに0が入っているとしてINC A(Aを+1する)命令を実行するとAレジスタは1になりますが，ゼロ・フラグは0になります．

なお，キャリ・フラグは，SCF/CCF命令で直接セットしたり反転したりすることができますが，ゼロ・フラグに対してはこのような特別な命令は用意されていません．

表1にキャリ・フラグとゼロ・フラグの動作について示します．

● サイン・フラグ(S)

これは符号(+/−)の状態を示すフラグです．

当然，このフラグは符号付きの演算をしている場合に使われます．符号付きとは2の補数の表現をした値のことです．8ビットのある演算結果を例として示すと，**表2**のようになります．

このように符号付き表現では最上位ビットが符号を示します．サイン・フラグは，0で正を，1で負を示します．つまりサイン・フラグとは，演算結果の最上位ビットをそのままフラグにコピーしたものと考えることもできます．

● パリティ/オーバ・フロー・フラグ(P/V)

これは1になっているビット数が偶数個あるか奇数個あるかのパリティを示したり，オーバ・フローを示すフラグです．命令によってどちらの意味になるかが決められています．

論理演算命令(AND/OR/XOR/ローテイト・シフト/IN r,(C)/DAAなど)の場合は，パリティ・フラグとしての意味をもちます．

算術演算命令(ADD/ADC/SUB/SBC/INC/DECなど)の場合は，オーバ・フロー・フラグとしての意味をもちます．

このフラグの示すオーバ・フローとは，符号付きの演算をしている場合のことです．符号なしの演算のオーバ・フローはキャリ・フラグを用います．

Z80のアドレッシング・モード

● アドレッシング・モードとは

アドレッシング・モードとは，CPUがある命令を実行する場合のパラメータ構造について規定したものです．ここではそのアドレッシング・モードについて，一つ一つ簡潔に説明していきます．**表3**に各種アドレッシング・モードをまとめます．

● アドレッシング・モードのいろいろ

▶ イミディエイト・モード (Immediate)

このモードは，その命令が対象とする1バイトのオペランドをもつ場合のアドレッシング・モードです．

イミディエイト・モードなどと難しそうな名前がついていますが，はやい話が指定した値をそのままレジスタやメモリに代入することです．

▶ 拡張イミディエイト・モード (Extended Immediate)

このモードはその名のとおりイミディエイト・モードの拡張で，1バイトではなく2バイトのオペランドをもつ場合のアドレッシング・モードです．

イミディエイト・モードとは8ビットで，拡張イミディエイト・モードが16ビットと覚えましょう．

▶ 拡張アドレス・モード (Extended)

これは2バイトからなるオペランド・アドレスをもつアドレッシング・モードです．指定したアドレスの1バイトあるいは2バイトのデータに対して操作をするモードです．

▶ インデックス・モード (Index)

これはインデックス・レジスタ(IX，IY)を用いる場合に使われます．

インデックス・モードでは，インデックス・レジスタ(IX，IY)の値とディスプレースメント(+d/−d)の和が使用されます．

ディスプレースメントとはLD A，(IX+d)のdのことです．このdは−128(80h)から+127(7Fh)の範囲で指定可能です．

▶ 間接アドレッシング・モード (Indirect)

Z80の場合にはHL，DE，BCなどペア・レジスタによりメモリ・アドレスを指定することをいいます．

例えばLD A,(HL)などがその代表です．これはHLレジスタで示されるアドレスの内容をAレジスタに代入するという命令です．

▶ レジスタ・モード (Register)

レジスタからレジスタへのデータ転送や演算など，レジスタどうしの操作の場合のモードです．これはオペコードの中に対象となるレジスタを指定するビットをもつ命令に適用されます．

▶ インプライド・モード (Implied)

〈表3〉アドレッシング・モード一覧

アドレッシング・モード	機械語コード(16進数)	アセンブリ言語
イミディエイト・モード	3E 01 06 CB	LD A,1 LD B,0CBH
拡張イミディエイト・モード	01 34 12 DD 21 AB 89	LD BC,1234H LD IX,89ABH
拡張アドレス・モード	3A 00 80 2A 34 12	LD A, (8000H) LD HL, (1234H)
インデックス・モード	DD 7E 60 FD 36 FF 99	LD A, (IX+56H) LD (IX−1), 99H
間接アドレッシング・モード	77 ED 40	LD A, (HL) IN B, (C)
レジスタ・モード	41 C5	LD B, C PUSH BC
インプライド・モード	C6 50 90	ADD A, 50H SUB B
ビット・モード	CB D7 CB 81 CB 7B	SET 2, A RES 0, C BIT 7, E
相対アドレス・モード	18 FD	JR LOOP1
ページ・ゼロ・モード	C7 FF	RST 00H RST 38H

Z80の8ビット演算命令(ADD/SUB/AND/OR/XOR/CP)では，Aレジスタの値と何らかのデータという形で必ずAレジスタが使われ，演算結果も常にAレジスタに格納されます．よってこれらの命令には，Aレジスタのオペランド指定がなくても，Aレジスタを使うことがわかりきっています．

このAレジスタのオペランド指定が省略されたモードが，インプライド・モードです．

しかし，ADD，ADC，SBC命令だけは16ビット演算との区別のために，ADD A,Bなどのように Aレジスタの記述が必要です．

▶ ビット・モード (Bit)

これはBIT/SET/RES命令がもつアドレッシング・モードです．これらの命令は，何レジスタの何ビット目をセット(1)するか，リセット(0)するか，または0か1かをチェックする命令です．

▶ 相対アドレス・モード (Relative)

これはJR/DJNZ命令だけがもつアドレッシング・モードです．JR/DJNZ(相対ジャンプ)命令は，JP(絶対ジャンプ)命令と異なり，ジャンプ先を現在位置からいくつ先というように相対的に表現します．

1バイトの相対アドレスの範囲は−128(80h)から+127(7Fh)で，当然この範囲を越えるような相対ジャンプはできません．

ただしアセンブリ言語で記述するときには，ジャンプ先が現在のロケーション・カウンタからどれだけ足した，または引いたアドレスかなどと意識する必要はありません．通常のJP命令と同様にジャンプ先のラ

ベルを指定するだけです．この相対アドレスの計算はアセンブラがやってくれます．

もしそのジャンプ先が−128〜＋127の範囲に入らない所にあった場合はアセンブラがエラーを出してくれるので，そのときにJP命令に直せばよいと気楽に考えて大丈夫です．

▶ ページ・ゼロ・モード(Page Zero)

これはRST命令のみがもつアドレッシング・モードです．

ページ・ゼロとは0000h〜00FFhの100hバイトの領域のことで，RST命令はオペランドに00H，08H，10H，18H，20H，28H，30H，38Hの8通りを受け入れます．

RST命令は1バイト命令のため機械語では，これらのアドレスを1バイト中の3ビット(ビット3〜5)で表現しています．

● **アドレッシング・モードのまとめ**

以上，アドレッシング・モードについて説明しましたが，細かいことにこだわらずに別の表現で要約してみると以下のようになります．

▶ 直接値をレジスタに代入するときはそのまま値を書く

例：LD HL,1234H　HLレジスタに1234hを代入

▶ ( )で囲まれたものは，そのアドレスのデータの意味

例：LD HL,(1234H)　Lレジスタに1234hのアドレスのデータを，Hレジスタに1235hのアドレスのデータを代入

例：LD A,(DE)　AレジスタにDEレジスタの値で示されるアドレスのデータを代入

▶ インデックス・レジスタ(IX/IY)は，(IX＋nnn)もしくは(IX−nnn)の形で−128〜＋127の範囲をアクセスできる．

例：LD A,(IX+5)　IXレジスタの値に5を足したアドレスのデータをAレジスタに代入

よく使うZ80の命令

Z80には158種類もの命令がありますが，これをすべて覚えておく必要はありません．その中でもよく使う命令は限られています．

ここでは紙面の都合もあり，特によく使う命令について解説します．これだけ覚えておけば，ほとんどのZ80のプログラムは大丈夫でしょう．

● **データの転送(リスト1)**

Z80ではデータの転送は，レジスタ-レジスタでも，レジスタ-メモリでも，すべてLD命令を使います．また代入方向は，LD A，Bなら，Bレジスタの内容をAレジスタに代入するというように，右から左です．

〈リスト1〉データ転送命令の例

```
LD      E,B          ;BレジスタからEレジスタへ
LD      A,(HL)       ;HLのアドレスのメモリをAレジスタへ
LD      (BC),A       ;AレジスタをBCのアドレスへ
LD      A,(IX+100)   ;IX+100のアドレスのメモリをAレジスタへ
PUSH    HL           ;HLレジスタのデータをスタックに積む
POP     DE           ;スタックからDEレジスタにデータを取り出す
LD      HL,8000H     ;HLレジスタに8000Hを代入
LD      DE,9000H     ;DEレジスタに9000Hを代入
LD      BC,1000H     ;BCレジスタに1000Hを代入
LDIR                 ;ブロック転送命令
EXX                  ;BC,DE,HLレジスタとBC',DE',HL'レジスタ交換
EX      AF,AF'       ;A,FレジスタとA',F'レジスタを交換
EX      DE,HL        ;DEレジスタとHLレジスタを交換
```

▶ レジスタ間転送

汎用8ビット・レジスタ間の転送はどのレジスタとの組み合わせでも可能です．

16ビット・レジスタ間の転送命令は，LD SP,HL/LD SP,IX/LD SP,IYの三つしかありません．汎用16ビット・レジスタの転送は，ペア・レジスタを8ビットずつに分けて2命令で実行します．

またインデックス・レジスタのように8ビットに分割できないレジスタは，PUSH命令でスタックに積んでからPOP命令で読み出します．

▶ レジスタ-メモリ間

8ビットならAレジスタで，16ビットならHL，IX，IYレジスタで使えます．また間接アドレッシング・モードを使えば，BCレジスタやDEレジスタで示されるアドレスのメモリ内容をAレジスタに転送することもできます．

▶ PUSH/POP命令

すでに説明したように，スタック領域に値を一時的に待避するときに使用する16ビット専用命令です．Aレジスタの待避はフラグ・レジスタと一緒にAFという形で行います．

またPUSH HLを実行したあとにPOP DEを実行すると，HLレジスタの内容をDEレジスタに転送することにもなります．

▶ ブロック転送命令

HLレジスタを転送元の先頭アドレスに，DEレジスタを転送先の先頭アドレスにして，BCレジスタで示されるバイト数だけ転送する命令が，LDIR命令です．1バイト転送するごとに，HLとDEレジスタは＋1し，BCレジスタは−1されます．

これとは逆に，転送元の終了アドレスをHLレジスタに，転送先の終了アドレスをDEレジスタにして，BCレジスタのバイト数だけ転送する命令がLDDR命令です．こちらは1バイト転送するごとに，HL，DE，BCレジスタを−1していきます．

▶ 交換命令

Z80には裏レジスタがあることは説明しました．こ

〈リスト2〉演算命令の例

```
INC   A        ;A =A +1
INC   BC       ;BC=BC+1
DEC   E        ;E =E -1
DEC   IY       ;IY=IY-1
ADD   A,B      ;A =A +B
ADD   HL,DE    ;HL=HL,DE
ADC   A,B      ;A =A +B +Cy(キャリ・フラグ)
ADC   HL,BC    ;HL=HL+BC+Cy(キャリ・フラグ)
SUB   D        ;A =A -D
SBC   A,E      ;A =A -E -Cy(キャリ・フラグ)
SBC   HL,DE    ;HL=HL-DE-Cy(キャリ・フラグ)
CP    B        ;AレジスタとBレジスタを比較
CP    55H      ;Aレジスタと55hを比較
AND   C        ;AレジスタとCレジスタをANDを取る
XOR   0AAH     ;Aレジスタと0AAhの排他的論理和を取る
```

の切り替えのためにEXX命令が用意されています．EXX命令はHL，DE，BCレジスタの切り替え用，EX AF，AF'命令がAレジスタとフラグ・レジスタの交換命令です．

さらにDEレジスタとHLレジスタの内容を切り替えるEX DE，HLという命令もあります．

また，交換命令そのものによるフラグの変化はありませんが，EX AF，AF'命令については，裏レジスタだったフラグ・レジスタに切り替わるわけですから注意が必要です．

● **演算命令(リスト2)**

▶ INC/DEC命令

レジスタの内容を＋1する命令がINC，－1する命令がDECです．汎用レジスタやインデックス・レジスタで使用できます．またフラグの変化も注意が必要で，8ビットではINC命令で0になっても，またDEC命令で255(0FFh)になってもキャリ・フラグは変化しません．また16ビット・レジスタのINC/DEC命令では，ゼロ・フラグも変化しません．

▶ ADD命令

Z80ではINC/DEC命令以外の演算命令は，8ビットならAレジスタに，16ビットならHLレジスタに集中しています．つまり，一度Aレジスタに数値を入れてから計算する必要があります．

まず加算命令としてはADD命令があります．8ビットならAレジスタと，16ビットならHLレジスタ，もしくはインデックス・レジスタとの足し算になります．桁あふれがあるとキャリ・フラグが1になります．

▶ ADC命令

これも加算命令なのですが，キャリ・フラグの内容も加算する命令です．命令実行前のキャリ・フラグが1のとき，足し算結果にさらに1が加算されます．実行前のキャリ・フラグがゼロなら，ADD命令と同じ演算結果になります．フラグの変化もADD命令と同じです．

▶ SUB命令

〈リスト3〉分岐命令の例

```
JP     8000H    ;8000h番地にジャンプ
JR     9800H    ;8000h番地にジャンプ(相対ジャンプ)
JP     C,0200H  ;キャリ・フラグが1なら200hにジャンプ
JR     NC,0100H ;キャリ・フラグが0なら100hにジャンプ
JP     NZ,4000H ;ゼロ・フラグが0なら4000hにジャンプ
JR     Z,4100H  ;ゼロ・フラグが1なら4100hにジャンプ
CALL   0800H    ;サブルーチン 800hをコール
....
RET             ;リターン
```

8ビット専用命令で，Aレジスタからオペランドの内容を引き算します．結果が0ならゼロ・フラグが1に，オペランドの値よりAレジスタのほうが小さければキャリ・フラグが1になります．

▶ SBC命令

これはADD命令に対するADC命令と同じで，この命令の実行前のキャリ・フラグが1であれば，引き算した結果からさらに1を引きます．フラグの変化はSUB命令と同じです．実行前のキャリ・フラグがゼロならSUB命令と同じ演算結果になります．

16ビットの引き算はこの命令しかないので，実行前にキャリ・フラグをリセットしてからこの命令を実行します．

▶ CP命令

8ビット専用の引き算命令ですが，実際に引き算した値はAレジスタには格納されません．フラグだけが変化します．フラグの変化はSUB命令と同じです．これは値を比較したいときに使用します．

16ビット・レジスタの比較をしたいときは，いったんレジスタを保存してから，キャリ・フラグをリセットしてSBC命令を実行するしかありません．

▶ AND，OR，XOR命令

論理演算命令で，8ビット専用です．論理演算命令では，キャリ・フラグはリセットされます．

● **分岐命令(リスト3)**

▶ JP/JR命令

プログラムの実行の流れを変える無条件分岐命令です．フラグは変化しません．JPは絶対アドレス，JRは相対アドレス・ジャンプ命令です．

▶ JP C,???/JR C,???/JP NC,??/JR NC,??命令

条件分岐命令です．キャリ・フラグが1ならオペランドのアドレスにジャンプします．0ならジャンプせず次の命令を実行します．NCがつくとフラグが0ならジャンプです．

▶ JP Z,???/JR Z,???/JP NZ,??/JR NZ,??命令

条件分岐命令で，ゼロ・フラグが1ならオペランドのアドレスにジャンプします．0ならジャンプせず次の命令を実行します．NZがつくとフラグが0ならジャンプします．

▶ CALL/RET 命令

サブルーチン・コール命令です．サブルーチンから戻るには RET 命令を使います．

▶ DJNZ 命令

プログラムにおいてループ処理はよく出てくる処理です．この命令は B レジスタをループのカウンタとして使う命令で，B レジスタを－1 して，0 以外なら指定アドレスにジャンプし，0 なら次の命令を実行します．フラグは変化しません．

この命令は相対アドレス・ジャンプになるので，ジャンプ先が－128～＋127 バイトの範囲外のときは使えません．

● 入出力命令(リスト 4)

Z80 にはメモリ空間と I/O 空間があることは説明しました．LD 命令はメモリ空間に対するアクセスでした．I/O 空間に対するアクセスには次の命令が用意されています．

<リスト 4> 入出力命令の例

```
IN      A,(80H)    ;I/Oアドレス80hのデータをAレジスタに入力
IN      B,(C)      ;CレジスタのI/OアドレスからBレジスタに入力
OUT     (90H),A    ;I/Oアドレス90hにAレジスタを出力
OUT     (C),E      ;CレジスタのI/OアドレスにEレジスタを出力
LD      HL,9000H   ;HLレジスタに9000hを代入
LD      B,16       ;Bレジスタに16を代入
LD      C,1CH      ;Cレジスタに1Chを代入
INIR               ;ブロック入力(16バイト I/Oアドレス1Ch)
LD      HL,8000H   ;HLレジスタに8000hを代入
LD      B,32       ;Bレジスタに32を代入
OTIR               ;ブロック出力(32バイト I/Oアドレス1Ch)
```

▶ IN 命令

オペランドで指定された I/O アドレスからデータを読み込み，A レジスタに転送します．また入力に 8 ビット汎用レジスタを使いたいときは，C レジスタを I/O アドレスとして，IN D,（C)のようにも使えます．

Z80七不思議　レジスタ・命令編

Q：EX DE, HL はあるのに EX BC, HL がないのはなぜ？

A：これは単に 8080A で EX DE, HL に相当する命令はある(XCHG)が，EX BC, HL に相当する命令がない，ということではないでしょうか．

Q：なぜ R レジスタが 7 ビットなのか．

A：Z80 CPU が開発された当時は，1K ビットの DRAM がやっと(？)世に出たような時代で，現在のような 4M や 16M ビットの DRAM が出てくるなどというのは想像のはるか彼方の世界でした．こんな時代でしたから，DRAM のリフレッシュ・アドレスは 7 ビットもあれば十分と考えたのだと想像されます．ちなみに，R レジスタが自動的に更新される部分は 7 ビットですが，レジスタ自体は 8 ビットの幅があります．下位 7 ビットの部分は自動的にインクリメントされますが，最上位ビットは書き込んだデータがそのままの状態で残ります．

また現在では Z80 に DRAM を使うことはまずありません．よって R レジスタの値を書き換えても動作に支障はありません．そこでもう一つおもしろい(？)応用例があります．R レジスタはリフレッシュ・サイクルを実行するたびに＋1 されるので，M1 サイクルのカウンタともいえます．つまり何命令実行したかの命令カウンタとしても使えるわけです．

Q：IXL, IYH などの命令はセカンド・ソースの Z80 でも使える公然の未定義命令になっている．未定義にしたのは何か意味があるのか(IX，IY はあくまでアドレス・ポインタ用として使ってほしいからか？)．

A：Z80 CPU は，前述のようにトランジスタの数を減らすべく，マイクロ・コードを使用せず，ハードワイヤ・ロジックで組まれました．このため，設計仕様にない命令が偶然できてしまいました．このようにしてできた命令群，例えば IX，IY レジスタの半分，8 ビットを操作する命令などは，その生い立ちからも未定義になっており，また出荷時にテストも行っておりません．

なお，Z80CPU と命令の上位互換性を持つ CPU，例えば Z380MPU では，これらの Z80CPU での未定義命令群は公式にサポートされています．

Q：掛け算，割り算命令がない．

A：まず掛け算や割り算を使用しなければならないアプリケーションというものは(特に制御ソフトウェアでは)あまりないということと，Z80CPU が開発された当時，掛け算や割り算をインプリメントするだけのトランジスタの贅沢ができなかったということによると思います．

<信垣育司>

〈リスト5〉ビット操作命令の例

```
SET  1,A          ;Aレジスタのビット1をセット
SET  7,(IX+0)     ;IXレジスタのメモリのビット7をセット
RES  3,E          ;Eレジスタのビット3をリセット
RES  5,(HL)       ;HLレジスタのメモリのビット5をリセット
BIT  2,H          ;Hレジスタのビット2をチェック
BIT  4,(IY-100)   ;IY-100のアドレスのメモリのビット4をチェック
```

▶ OUT命令

IN命令とは逆に指定されたI/OアドレスにAレジスタの値を出力する命令です．これも同様に，Cレジスタによる間接アドレッシングによって8ビット汎用レジスタの値を出力することも可能です．

▶ ブロック入出力命令

メモリ-メモリ間ブロック転送にLDIRやLDDR命令があったように，Z80ではI/O空間とメモリ空間に対しても，ブロック入出力命令があります．

INIR命令は，Cレジスタで示されるI/Oからデータを入力し，HLレジスタで示されるメモリに格納し，これをBレジスタのバイト数だけ繰り返します．1バイト入力するごとにHLレジスタは+1され，Bレジスタは−1されます．当然のことながらCレジスタは変化しません．

逆にOTIR命令は，HLレジスタで示されるメモリからデータを読み出し，Cレジスタで示されるI/Oに出力し，Bレジスタのバイト数だけ繰り返します．これも1バイト出力ごとにHLレジスタは+1，Bレジスタは−1されます．

● ビット操作命令(リスト5)

8ビット中の任意のビットに対するビット操作です．すべての汎用レジスタのすべてのビットに対して可能です．

▶ SET命令

オペランドで指定されたレジスタのビットをセット(1にする)します．フラグは変化しません．

▶ RES命令

オペランドで指定されたレジスタのビットをリセット(0にする)します．フラグは変化しません．

▶ BIT命令

オペランドで指定されたレジスタのビットの状態をチェックします．“1”ならゼロ・フラグが“0”，“0”ならゼロ・フラグが“1”です．反転することに注意してください．

● ローテイト/シフト命令(リスト6)

8ビット・レジスタの内容をローテイト/シフトする命令です．

ローテイト命令は，シフトされてはみ出たビット・データが戻って入力ビットとなります．シフト命令は，はみ出たビットはキャリ・フラグに入り，シフト入力には“0”が入ります．

〈リスト6〉ローテイト/シフト命令の例

```
RLC  B        ;Bレジスタを左にローテイトする
RL   (IX+0)   ;IXレジスタのメモリの内容を左にローテイトする
RRC  C        ;Cレジスタを右にローテイトする
RR   (IY+0)   ;IYレジスタのメモリの内容を右にローテイトする
SLA  A        ;Aレジスタを左にシフトする
SRL  B        ;Bレジスタを右にシフトする
```

〈リスト7〉CPU制御命令の例

```
NOP           ;何もしない命令
HALT          ;CPU動作停止命令
IM   2        ;割り込みモード2に設定
DI            ;/INT割り込み受け付け禁止命令
EI            ;/INT割り込み受け付け許可命令
```

▶ RLC/RL命令

左へローテイトする命令です．RLC命令は8ビット内で，RL命令はキャリ・フラグも含めた9ビット内でシフトします．どちらもキャリ・フラグにはオペランドの最上位ビットが入ります．

▶ RRC/RR命令

右へローテイトする命令です．RRC命令は8ビット内で，RR命令はキャリ・フラグも含めた9ビット内でシフトします．どちらもキャリ・フラグにはオペランドの最下位ビットが入ります．

▶ SLA/SRL命令

SLA命令はオペランドの内容を左にシフトし，最上位ビットをキャリ・フラグに入れます．

SRL命令はオペランドの内容を右にシフトし，最下位ビットをキャリ・フラグに入れます．

● CPU制御命令(リスト7)

▶ NOP命令

何もしない命令です．無駄な命令のように思えますが，制御系では微妙なタイミングを取ったりするときに使用されます．

▶ HALT命令

CPUを停止させる命令です．割り込みが入るまで，CPUは動作を停止します．

▶ IM/DI/EI命令

割り込み関連の命令です．割り込みについては第6章を参照してください．

最後に，Z80の命令を整理してまとめたものを表4に示します．

◈引用文献◈

(1) 神崎康宏；トランジスタ技術SPECIAL No.6，Z80ハード＆ソフトのすべて，p.24.

(トランジスタ技術1994年6月号に加筆，修正)

〈表 4〉[1] Z80 の命令一覧（□：オペランドのない命令）

●転送命令

ロード命令	
LD	レジスタ，メモリ間のデータ転送(8ビット，16ビット)
PUSH	レジスタの内容をスタックへプッシュする(16ビット)
POP	スタックの内容をレジスタへポップする(16ビット)
交換命令	
EX	レジスタ・ペア同士の内容をそれぞれ交換する
EXX	レジスタ・ペアBC，DE，HLの内容をBC′，DE′，HL′の内容と交換する
ブロック転送命令	
LDI	(DE)←(HL)，DE←DE+1，HL←HL+1，BC←BC−1
LDIR	(DE)←(HL)，DE←DE+1，HL←HL+1，BC←BC−1，BC=0まで繰り返す
LDD	(DE)←(HL)，DE←DE−1，HL←HL−1，BC←BC−1
LDDR	(DE)←(HL)，DE←DE−1，HL←HL−1，BC←BC−1，BC=0まで繰り返す
CPI	A−(HL)，HL←HL+1，BC←BC−1
CPIR	A−(HL)，HL←HL+1，BC←BC−1，A=0またはBC=0まで繰り返す
CPD	A−(HL)，HL←HL−1，BC←BC−1
CPDR	A−(HL)，HL←HL−1，BC←BC−1，A=0またはBC=0まで繰り返す

●演算命令

加減算	
ADD	キャリを含まない加算(8ビット，16ビット)
ADC	キャリを含む加算　(8ビット，16ビット)
SUB	キャリを含まない減算(8ビット)
SBC	キャリを含む減算　(8ビット，16ビット)
CP	比較(フラグのみ変化する)(8ビット)
INC	+1する(8ビット，16ビット)
DEC	−1する(8ビット，16ビット)
論理演算	
AND	アキュムレータとの論理積をとる(8ビット)
OR	アキュムレータとの論理和をとる(8ビット)
XOR	アキュムレータとの排他的論理和をとる(8ビット)
汎用算術演算	
DAA	アキュムレータの10進補正
CPL	アキュムレータの内容の1の補数をとる
NEG	アキュムレータの内容の2の補数をとる
CCF	キャリ・フラグのビットを反転する
SCF	キャリ・フラグをセットする

●CPU制御命令

NOP	プログラム・カウンタを1進める
HALT	CPUを停止させる
DI	INT割り込みを無効にする(IFF_1，IFF_2をリセットする)
EI	INT割り込みを有効にする(IFF_1，IFF_2をセットする)
IM	割り込みモードを0〜2にセットする

●ローテイト，シフト命令

ローテイト	
RLCA	アキュムレータの内容を左へローテイトする
RLA	アキュムレータの内容を左へローテイトする
RRCA	アキュムレータの内容を右にローテイトする
RRA	アキュムレータの内容を右にローテイトする
RLC	オペランドの内容を左へローテイトする
RL	オペランドの内容を左へローテイトする
RRC	オペランドの内容を右にローテイトする
RR	オペランドの内容を右にローテイトする
シフト	
SLA	オペランドの内容を左へシフトする
SRA	オペランドの内容を右にシフトする
SRL	オペランドの内容を右にシフトする
RLD	
RRD	

●ビット操作命令

BIT	オペランドで指定されたビットの反転した値をZフラグに入れる
SET	オペランドで指定されたビットをセットする
RES	オペランドで指定されたビットをリセットする

●分岐命令

ジャンプ命令	
JP	オペランドの内容をPCにロードし，次の命令はそこから始まる．無条件と条件付きがある
JR	オペランドで指定されたディスプレースメントだけ離れた番地へジャンプする．条件つきもある
DJNZ	Bレジスタの内容を−1し，0でなければJRと同様にジャンプし，0ならば次の命令を実行する
コール，リターン命令	
CALL	コール命令の次の命令の番地をスタックにプッシュした後，オペランドで示された番地へ飛ぶ．条件付きもある
RET	スタックからコール命令の次の命令のアドレスをPCにポップして元のプログラムにもどる．条件付きもある．
RET(条件付き)	
RETI	割り込みサービス・ルーチンの終了時に使用するRET命令
RETN	ノンマスカブル割り込みに対するサービス終了時に使用するRET命令

●入出力命令

入力命令	
IN	それぞれオペランドで指定されたポートからレジスタへデータを入力する
INI	(HL)←(C)，B←B−1，HL←HL+1
INIR	(HL)←(C)，B←B−1，HL←HL+1，B=0まで繰り返す
IND	(HL)←(C)，B←B−1，HL←HL−1
INDR	(HL)←(C)，B←B−1，HL←HL−1，B=0まで繰り返す
出力命令	
OUT	それぞれオペランドで指定されたポートへレジスタからデータを出力する
OUTI	(C)←(HL)，B←B−1，HL←HL+1
OTIR	(C)←(HL)，B←B−1，HL←HL+1，B=0まで繰り返す
OUTD	(C)←(HL)，B←B−1，HL←HL−1
OTDR	(C)←(HL)，B←B−1，HL←HL−1，B=0まで繰り返す

アクセス・タイミングの計算からバンク切り替えまで

メモリ周辺回路の設計

村田浩義/北川信孝

メモリとは何だろう

● 高速大容量化の進むメモリ

半導体技術の進歩でメモリの大容量化が進み，パソコンに数十Mバイトのメモリを積むことも珍しくなくなってきました．

いまやZ80マイコンの世界でも，メモリ容量32Kバイトはかなり小さいほうです．Z80マイコン・システムなら256Kビット・メモリを2個つんで，64Kバイト・フル・メモリ・システムを構成できます．

メモリの高速化，大容量化は日進月歩ですが，逆にZ80などで多用されていた64Kビットのメモリは入手しづらくなってきています．このままでは256Kビットのメモリも心配です．M(メガ)バイト時代になっても，Z80のような組み込み機器用のために，せめて256Kビットのメモリだけは作り続けていただきたいものです．

ここではまずメモリとは何か，どんなものがあるか見てみましょう．

● メモリとは

メモリは"1"か"0"かという情報を記憶する素子です．汎用ロジックICの中ではDフリップフロップ(HC74)もメモリとして考えることができます(図1)．Dフリップフロップを複数個並べて，アドレス・バスによって一つだけを選択する回路と，データ・バスにデータを入出力する回路，そして読み出しか書き込みかのコントロール回路を設ければ，りっぱにメモリを構成できます．

〈図1〉Dフリップフロップを使った1ビットの記憶

このように，アドレスを指定してデータを読み書きできる素子がメモリです(図2)．しかしDフリップフロップを並べていたのでは効率が悪いので，メモリ素子として特化したICが作られています．マイコン・システムでは，特に断らない限り半導体メモリのことを略してメモリと呼んでいます．

メモリを大きく分けると，揮発性メモリと不揮発性メモリに分類できます．

揮発性メモリは，DRAM(Dynamic RAM)やSRAM(Static RAM)に代表されるような，電源を切るとデータが失われてしまうメモリのことです．読み書きやランダム・アクセスが自在にできるので，RAM(Random Access Memory)とも呼びます．

不揮発性メモリは，UV EPROM(UltraViolet Eraseable Programmable ROM)に代表される，電源を切ってもデータが失われないメモリです．基本的に

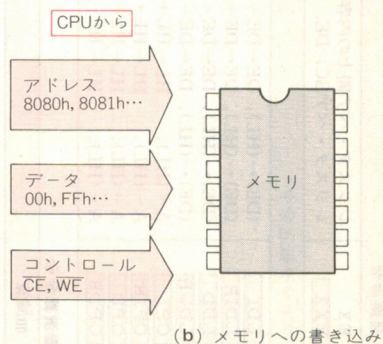
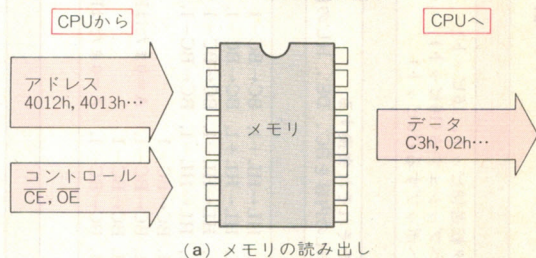
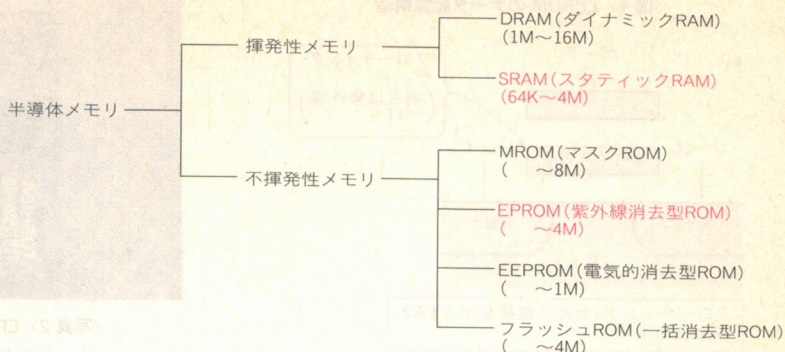
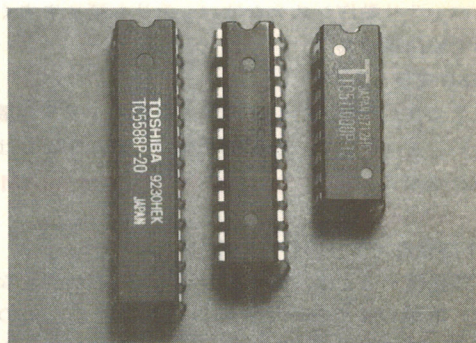
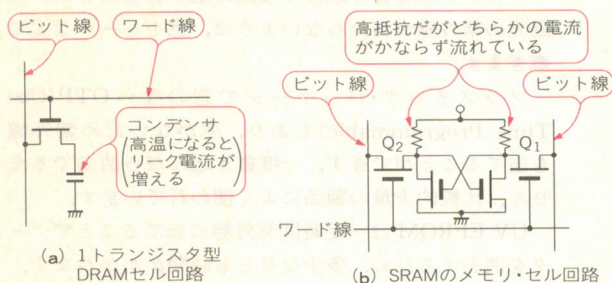
〈図2〉メモリの読み書き

(a) メモリの読み出し

(b) メモリへの書き込み

〈図 3〉SRAM と DRAM のデータ記憶構造

(a) 1トランジスタ型 DRAMセル回路

(b) SRAMのメモリ・セル回路

〈写真 1〉SRAM と DRAM の外観(左から 64 K ビット SRAM, 8 K ビット SRAM, 1 M ビット DRAM)

書き込みができず読み出し専用なので，ROM (Read Only Memory)とも呼びます．

表 1 にメモリの分類を示します．

● 揮発性メモリのいろいろ

電源を切ると内容が失われるわけですから，電源 ON ですぐに動作させる組み込みマイコンでは，RAM にプログラムを記憶させておくことはできません．RAM はデータ領域やスタック・エリアとして使われます．

揮発性メモリはさらに DRAM と SRAM に分けられます．**図 3** に DRAM と SRAM の内部構造図を，**写真 1** に外観を示します．

▶ DRAM とは

DRAM は“0”，“1”の記憶部分がコンデンサにより形成されており，このコンデンサに電荷を充電することで“0”，“1”の情報を記憶します．記憶部分がコンデンサのため，リーク電流により電荷が減少してしまい，ほおっておくと内容が消えてしまいます．

このため，一定時間ごとに一度内容を読み出し再書き込みをします．この作業をリフレッシュと呼びます．またこのような構造のため，高温ではアクセス・タイムも遅くなり，リーク電流も増加します．

また，例えば1M ビット DRAM はアドレス・ピンが 10 本と，1 M(メガ) のアドレスを指定するのに半分の本数しかありません．これはアドレスを時分割して指定することでピン数を減らし，実装面積を小さくできるようになっているからです．

この時分割アドレス指定のため，$\overline{\mathrm{RAS}}$ と $\overline{\mathrm{CAS}}$ によるアドレス制御が必要で，なかなか使いにくいメモリです．

DRAM はパソコンなどでは主メモリとして使われていますが，8 ビット程度のマイコン・システムでは現在ではあまり使われません．

▶ SRAM とは

SRAM はまさに，D フリップフロップを並べたメモリと考えることができます．フリップフロップにより“0”，“1”を記憶しているため，DRAM のリフレッシュのような処理は必要ありません．

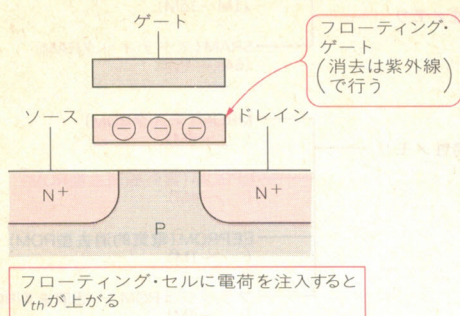
また $\overline{\mathrm{RAS}}$ や $\overline{\mathrm{CAS}}$ のような時分割アドレス指定ではなく，そのままアドレスを指定できるので非常に使いやすいメモリです．

データ・バスも 8 本のものが主流で，8 ビット CPU である Z80 などに最適です．またサイクル・タイムとアクセス・タイムが同じなので，高速クロック回路でも設計しやすく，使いやすいメモリです．

ただし DRAM と同様に，高温でリーク電流が増加するので，バッテリ・バックアップする場合は使用温度に留意してバッテリを決める必要があります．

逆に，低温時に MOS IC は消費電流が増加するの

〈図4〉 EPROMのデータ記憶構造

〈写真2〉 EPROMの外観
(上：ワンタイムPROM，下：UV EPROM)

で，バッテリ動作システムはバッテリも減りやすくなるので要注意です．

● Z80で使うRAM

以上より，DRAMはMバイト単位でメモリ容量が欲しいときに最適で，Z80程度のシステムではもはや使われません．Z80で使うRAMとしてはSRAMを使うのが一般的です．

● 不揮発性メモリのいろいろ

不揮発性メモリは，電源を切っても内容が消えないので，電源ONで動作する組み込み機器のためのプログラムや初期データ用のメモリとして使われます．

不揮発性メモリにはIC製造時に内容が決まってしまうマスクROMと，ユーザ側でも専用回路によりデータを書き込むことのできるEPROMとに分けられます．

さらにEPROMにはデータの消去方法の違いにより，紫外線で消去が可能なUV EPROM，電気的に消去が可能なEEPROM，そして最近使われるようになってきたEEPROMの親戚でもある一括消去型のフラッシュROMに分けられます．

図4にEPROMの構造図を，**写真2**にEPROMの外観を示します．

▶ マスクROMとは

これはIC製造時にマスク(ICを作るときのフィルム)で内容が決まってしまうメモリで，大量生産品にしか使いません．また内容の変更もできないので開発時に使うこともありません．

▶ EPROMとは

このメモリに内容を書き込むには，専用の書き込み器(ROMライタ)が必要です．この書き込み器で，フローティング・ゲートに電荷を注入することによりデータを書き込みます．

EPROMの中でも開発時などにいちばん使われるのがUV EPROMです．UV EPROMはセラミック・パッケージで，紫外線を照射するための窓がパッケージの上部に開いていてチップが見える構造になっています．紫外線を一定時間照射することによりデータを消去(フローティング・ゲートの電荷を抜く)できます．

プログラムを書き込んで製品に組み込むときは，太陽光の紫外線が当たらないように，窓をシールなどで塞ぎます．

プラスチック・パッケージで窓のないOTP(One Time Programmable)もあり，窓がないため紫外線を当てることができず，一度書き込んだら消去できません．比較的少量の製品によく使われています．

UV EPROMは一定時間紫外線に当てることでデータを消去するため，多少なりとも時間がかかります．EEPROMは電気的にデータを消去するので，消去して書き込みを終えるまでの時間が少なくてすみます．フラッシュROMに至っては瞬間的に全データを消去できるようになっています．

EPROMは，アドレスの指定がSRAMと同じで，データ・バス幅も8ビットと，Z80とも簡単に接続することができる使いやすいメモリです．

● Z80に使うROM

大量生産するものならまだしも，少量生産品ではマスクROMは論外です．やはり書き込み器などツールがそろっているUV EPROMが，Z80システムには最適でしょう．

● メモリのアクセス手順

ここではメモリからデータを読み出したり，データを書き込むときの制御の手順を説明します．特にZ80システム用に最適なSRAMとEPROMについて取り上げます．

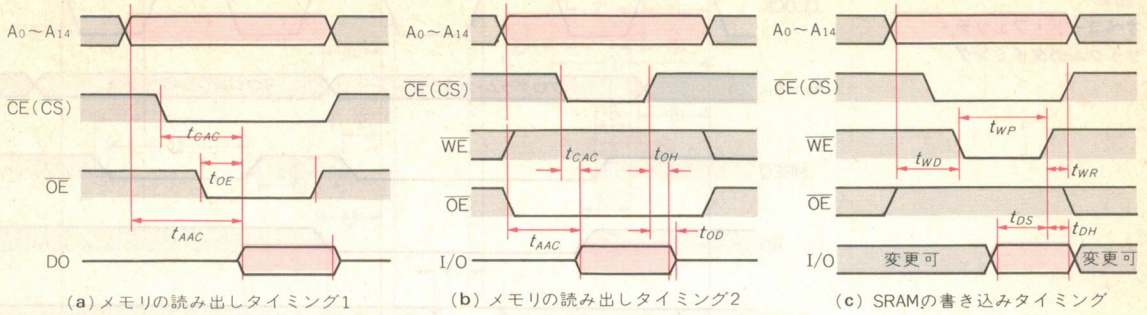
メモリのアクセスに必要な信号は，アドレス，チップ・セレクト($\overline{CE}$)，そして読み出しか書き込みかの指定である$\overline{OE}$や$\overline{WE}$です．

図5に一般的なメモリのアクセス方法と，**表2**に代表的な256Kビット・メモリのアクセス・タイムを示します．

アクセス・タイムとは，アドレスが確定してからデータが正しく出力されるまでの時間をいいます．

高速なクロックで動作するマイコンになればなるほど，このアクセス・タイムが速い必要があるのです．

〈図5〉EPROMとSRAMのアクセスのようす

(a) メモリの読み出しタイミング1　(b) メモリの読み出しタイミング2　(c) SRAMの書き込みタイミング

〈表2〉
代表的な256Kビット・メモリのアクセス・タイム

型名	スイッチング特性								メーカ
	t_{AAC} max (ns)	T_{CAC} max (ns)	t_{OE} max (ns)	t_{WP} min (ns)	t_{DS} min (ns)	t_{DH} min (ns)	t_{WD} min (ns)	t_{WR} min (ns)	
MB84256A-10	100	100	40	60	40	0	0	5	富士通
MB84256A-70	70	70	35	50	25	0	0	5	
MB84257-12	120	120		70	45	0	0	5	
HM62256P-12	120		60	90	50	0	0		日立
HM62256P-8	85		45	70	40	0	0		
HM62832HLP/JP-25	25	25	12	15	12	0	0	0	
μPD43256G/GU-10L	100	100	50	70	40	0	0	5	NEC
μPD43256G/GU-15L	150	150	70	90	60	0	0	5	
μPD43258ACR/LA-25	25	25	12	15	12	0	0	0	
TC55257APL-85	85	85	45	60	40	0	0	5	東芝
TC55257APL-10	100	100	50	70	40	0	0	5	
TC55257APL-12	120	120	60	80	50	0	0	5	

(a) SRAMのアクセス・タイム

型名	スイッチング特性			メーカ
	t_{AAC} max (ns)	t_{CAC} max (ns)	t_{OE} max (ns)	
MBM27C256A-20W	200	200	70	富士通
MBM27C256H-10	100	100	45	
MBM27C256H-12	120	120	50	
HN27C256AG-10	100	100	60	日立
HN27C256AG-15	150	150	70	
HN27C256HG-70	70	70	40	
μPD27C256AK-12	120	120		NEC
μPD27C256AK-15	150	150		
μPD27C256AK-20	200	200		
TC57256AD-12	120	120	60	東芝
TC57256AD-15	150	150	70	

(b) UV EPROMのアクセス・タイム

まずはアドレスを確定することが先決です．そしてチップ・セレクト($\overline{\mathrm{CE}}$)をアクティブにします．そして読み出しには$\overline{\mathrm{OE}}$をアクティブにしてデータ・バスにデータを出力させます．ROMもRAMもこれは同じです．RAMの場合はデータの読み出し時は$\overline{\mathrm{WE}}$をHレベルにしておきます．

また書き込みでは$\overline{\mathrm{WE}}$をアクティブにし，データ・バスに書き込むデータを出力します．そして$\overline{\mathrm{WE}}$クロックの立ち上がりでRAMはデータを取り込みます．このときは読み出し信号である$\overline{\mathrm{OE}}$をHレベルにしておきます．

これらの制御線は必要のないときはHレベルにしておきます．

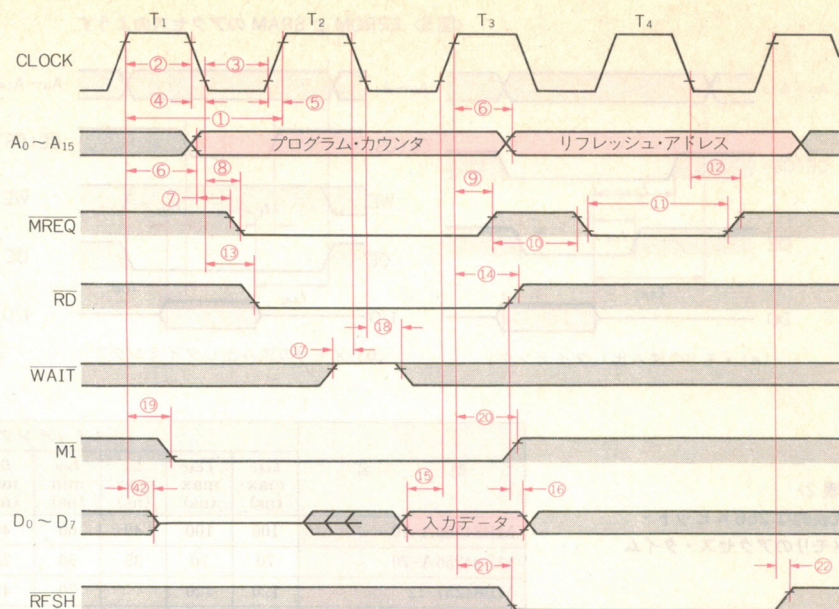
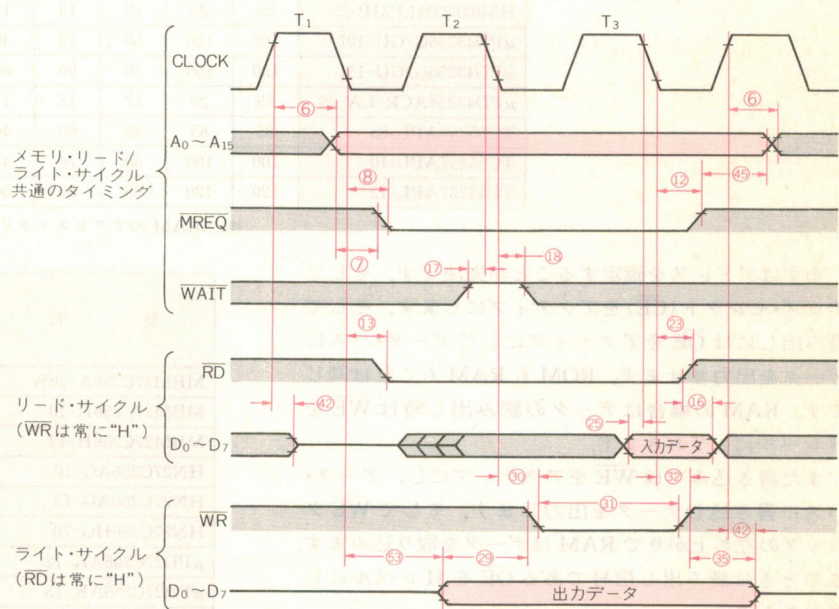
● $\overline{\mathrm{CE}}$や$\overline{\mathrm{OE}}$の制御タイミングのいろいろ

ここで，図5(a)，(b)のメモリの読み出しタイミングをみてください．この違いは，$\overline{\mathrm{CE}}$をアクティブにしてから$\overline{\mathrm{OE}}$をアクティブにしているか，先に$\overline{\mathrm{OE}}$をアクティブにしてから$\overline{\mathrm{CE}}$をアクティブにしているかの違いです．基本的にメモリの読み出しはどちらでも動作しますが，一般的には図(a)の方法でアクセスします．

また変則的な例として，ROMのアクセス速度をギリギリまで使うため，$\overline{\mathrm{CE}}$をグラウンドに落としたまま，$\overline{\mathrm{OE}}$だけで読み出し制御をする場合もあります．

表2をみるとわかりますが，アドレスが確定してからのアクセス時間t_{AAC}と，$\overline{\mathrm{CE}}$がイネーブルになってからのアクセス時間t_{CAC}は同じです．

ということは，アドレス・デコード回路からの出力を$\overline{\mathrm{CE}}$に接続した場合，アドレス・デコーダの遅延時

〈図6〉
オペコード・フェッチ・サイクルのタイミング

〈図7〉
メモリ・リード/ライト・サイクルのタイミング

間だけ，t_{AAC}によるアクセス時間より遅くなるわけです．

そこで，ROMの場合は読み出し動作だけなわけですから，アドレス・デコードにより読み出すアドレスが決まったら，これを$\overline{OE}$に接続してアクセス時間をかせぐ方法があるのです．$\overline{OE}$がアクティブになってからのアクセス時間t_{OE}は，t_{AAC}やt_{CAC}の半分程度と早いので，$\overline{CE}$をグラウンドに接続しておけば，アドレス・バスが確定した時点でアクセスが始まり，アドレス・デコーダでの遅延時間分をあまり気にする必要がなくなります．

RAMについては，書き込み動作があるため，この方法を使うのは少々大変です．

Z80のメモリ・アクセス・タイミング

● メモリ・アクセス・タイミングの重要性

マイコンは命令をメモリから読み込み，動作します．またその命令によって，メモリからデータを読み出したり，メモリにデータを書き込んだりします．

〈表 3〉 オペコード・フェッチとメモリ・リード/ライト・サイクル(単位 ns)

番号	記号	項目	6MHz 版		8MHz 版		10MHz 版		20MHz 版	
			最小	最大	最小	最大	最小	最大	最小	最大
1	t_{CC}	クロック周期	162	DC	125	DC	100	DC	50	DC
2	t_{wCh}	クロック"H"パルス幅	65	DC	55	DC	40	DC	20	DC
3	t_{wCl}	クロック"L"パルス幅	65	DC	55	DC	40	DC	20	DC
4	t_{fC}	クロック立ち下がり時間		20		10		10		10
5	t_{rC}	クロック立ち上がり時間		20		10		10		10
6	$t_{dCr(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
7	$t_{dA(MREQf)}$	$\overline{MREQ}$ に先立つアドレス出力確定時間	35		20		5		−15	
8	$t_{dCf(MREQf)}$	クロック立ち下がりから $\overline{MREQ}$="L"になるまでの遅延		70		60		55		40
9	$t_{dCr(MREQr)}$	クロック立ち下がりから $\overline{MREQ}$="H"になるまでの遅延		70		60		55		40
10	t_{wMREQh}	$\overline{MREQ}$"H"パルス幅	65		45		30		10	
11	t_{wMREQl}	$\overline{MREQ}$"L"パルス幅	132		100		75		25	
12	$t_{dCf(MREQr)}$	クロック立ち下がりから $\overline{MREQ}$="H"になるまでの遅延		70		60		55		40
13	$t_{dCf(RDf)}$	クロック立ち下がりから $\overline{RD}$="L"になるまでの遅延		80		70		65		40
14	$t_{dCr(RDr)}$	クロック立ち上がりから $\overline{RD}$="H"になるまでの遅延		70		60		55		40
15	$t_{sD(Cr)}$	クロック立ち上がりに対するデータ・セットアップ時間	30		30		25		12	
16	$t_{hD(RDr)}$	$\overline{RD}$ 立ち上がりに対するデータ・ホールド時間	0		0		0		0	
17	$t_{sWAIT(Cf)}$	クロック立ち下がりに対する $\overline{WAIT}$ セットアップ時間	60		50		20		7.5	
18	$t_{hWAIT(Cf)}$	クロック立ち下がり後の $\overline{WAIT}$ ホールド時間	10		10		10		10	
19	$t_{dCr(M1f)}$	クロック立ち上がりから $\overline{M1}$="L"になるまでの遅延		80		70		65		45
20	$t_{dCr(M1r)}$	クロック立ち上がりから $\overline{M1}$="H"になるまでの遅延		80		70		65		45
21	$t_{dCr(RFSHf)}$	クロック立ち上がりから $\overline{RFSH}$="L"になるまでの遅延		110		95		80		60
22	$t_{dCr(RFSHr)}$	クロック立ち上がりから $\overline{RFSH}$="H"になるまでの遅延		100		85		80		60
23	$t_{dCf(RDr)}$	クロック立ち下がりから $\overline{RD}$="H"になるまでの遅延		70		60		55		40
25	$t_{sD(Cf)}$	クロック立ち下がりに対するデータ・セットアップ時間 (M2, M3, M4, M5 サイクル時)	40		30		25		12	
29	$t_{dD(WRf)}$	$\overline{WR}$ 立ち下がりに先立つデータ確定時間	22		5		40		−10	
30	$t_{dCf(WRf)}$	クロック立ち下がりから $\overline{WR}$="L"になるまでの遅延		70		60		55		40
31	t_{wWR}	$\overline{WR}$ パルス幅	132		100		75		25	
32	$t_{dCf(WRr)}$	クロック立ち下がりから $\overline{WR}$="H"になるまでの遅延		70		60		55		40
35	$t_{dWRr(D)}$	$\overline{WR}$="H"になってからの出力データ保持時間	30		15		10		0	
42	$t_{dCr(Dz)}$	クロック立ち上がりからデータ・バス・フロート状態までの遅延		80		70		65		40
45	$t_{dCr(A)}$	$\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ または $\overline{WR}$ からのアドレス保持時間	35		20		20		0	
53	$t_{dCf(D)}$	クロック立ち下がりからデータ出力までの遅延		130		115		110		75

このメモリからの命令，データの読み出しやデータの書き込みには，それぞれ制御の方法やタイミングというものが決まっています．

この制御の方法やタイミングを守らないと，正しくメモリからデータを読み書きすることができません．このタイミングを守らないと，どんなデータが読み書きされるかわかったものではありません．CPU にしてみれば，読み込んだ命令が実は正しくないかもしれないという状況が起こり得るのです．

ここではまず，Z80 がどのようなタイミングでメモリを読み書きするかを説明します．オペコード・フェッチ・サイクルやメモリ・リード/ライト・サイクルについては第1章で説明しましたが，ここではさらに詳しく，図6と図7にそのタイミングを，表3にアクセス・タイムを示します．

● オペコード・フェッチ・サイクル

Z80 が命令を読み込むことを，オペコード・フェッチと呼び，このときの CPU の状態をオペコード・フェッチ・サイクルと呼びます．

Z80 の命令は，オペコードと呼ばれる動作そのものを指定するデータと，オペランドと呼ばれるレジスタやアドレスを指定するデータの組み合わせで構成されます．

命令によってバイト数が異なり，Z80 では最短バイト数の命令は1バイトで，最長バイト数の命令は4バイトで構成されています．

まず初めに CPU は，T_1の立ち上がりで PC(プログラム・カウンタ)の内容をアドレス・バスに出力し，$\overline{M1}$ を立ち下げます．そして T_1の立ち下がりで $\overline{MREQ}$ (メモリ要求信号)と $\overline{RD}$(読み出し要求)をアクティブ

にします．

これらの信号で選択されたメモリからデータが出力され，CPUはT_3の立ち上がりでデータを取り込みます．

そして次のT_3，T_4の二つのサイクルで，CPUはDRAMに対するリフレッシュ信号を出力し，内部では先ほど取り込んだデータ(オペコード)の解析(デコード)と実行がなされます．このとき出力されるリフレッシュ信号は，$\overline{\text{MREQ}}$と$\overline{\text{RFSH}}$信号がアクティブになり，リフレッシュ・アドレスがアドレス・バスに出力されます．

● オペコード・フェッチ・サイクルをメモリからみると

このようにオペコード・フェッチ・サイクルは，4ステート(クロック)サイクルなのですが，T_3とT_4ではアドレス・バスにリフレッシュ・アドレスが出力されるので，実質的な命令読み込みのためのアドレス出力は，T_1とT_2の時間だけです．

メモリにしてみれば，アドレスが確定したあとT_3の立ち上がりでCPUがオペコードを読み込む前までに，指定されたアドレスのデータ，つまりこの場合は命令をデータ・バスに出力しなければなりません．これがメモリが守るべきアクセス・タイミングです．

またオペコードがROM上に置かれようがRAM上に置かれようが，CPUはそのことについてまったく感知していないことを頭に入れてください．

たしかに電源投入時はプログラムはROM上にしかありませんが，その途中でRAM上にプログラムを展開して動作するというシステムも構成可能です．

つまりROMに対してもRAMに対しても，通常は同じタイミングでオペコード・フェッチが行われるわけです．

● メモリ・リード/ライト・タイミング

次はオペコードの読み出しではなく，オペランドやデータをメモリから読み込んだり，データを書き込んだりするときのタイミングをみてみます．Z80ではこれをメモリ・リード/ライト・サイクルと呼んでいます．

オペコード・フェッチがT_1とT_2の2クロック時間であったのに対して，メモリ・リード/ライト・サイクルはT_1からT_3の立ち下がりまでの2.5クロック時間です．オペランドの読み込み時はPCの内容が，メモリの読み込み時はそのメモリ・アドレスがアドレス・バスに出力されます．このとき，$\overline{\text{MREQ}}$と$\overline{\text{RD}}$がアクティブになり，メモリから出力されたデータをCPUはT_3の立ち下がりで取り込みます．

データの書き込み時は，メモリ・アドレスがアドレス・バスに出力され，$\overline{\text{MREQ}}$と$\overline{\text{WR}}$がアクティブになり，CPUからデータ・バスに書き込むデータが出力されます．

● メモリ・リード/ライト・サイクルをメモリからみると

オペコード・フェッチ・サイクルでは，T_3の立ち上がりでデータがCPUに取り込まれていたのが，メモリ・リード・サイクルでは，T_3の立ち下がりでデータがCPUに取り込まれるので，0.5クロック時間アクセスに余裕がある計算になります．

またメモリ・ライト・サイクルでは，$\overline{\text{MREQ}}$がアクティブである時間はメモリ・リード・サイクルと同じ2.5クロック時間ありますが，$\overline{\text{WR}}$がアクティブである時間はほぼ1クロック分で，T_2の立ち下がりでアクティブになります．

つまりデータの読み込みに関しては，オペコード・フェッチ・サイクルを満足できるのであれば問題はないが，書き込みに関しては$\overline{\text{WR}}$信号の立ち下がりのタイミングとその時間に注目する必要があるのです．またメモリへの書き込みのタイミングは$\overline{\text{WR}}$クロックの立ち上がりで行います．

またオペコード・フェッチ同様，メモリ・リード・サイクルもROMとRAMの両方に対して考えられ，また同じタイミングが要求されます．しかしメモリ・ライト・サイクルだけは，メモリへの書き込み動作ですから，RAMについてのみ考えればよいわけです．

Z80とメモリの接続

● マイコン・システムに必要なROMとRAM

Z80は電源を入れるとアドレス0000h番地から実行を開始します．したがって電源を切ってもプログラムを記憶しておくことのできるROMが必ず必要です．

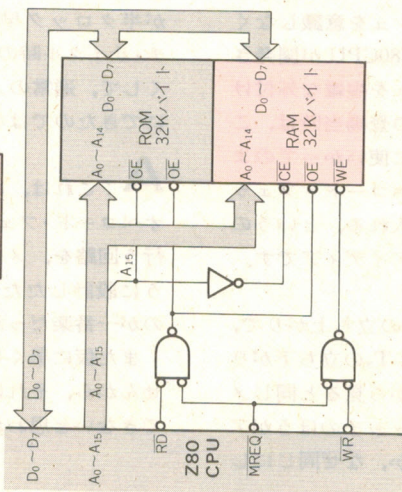
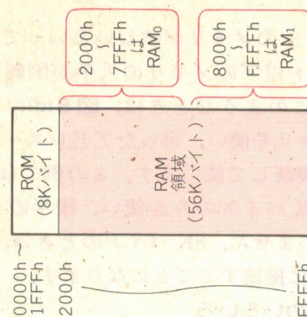
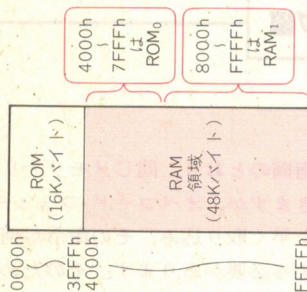
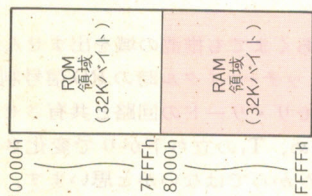
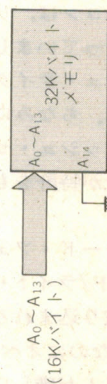
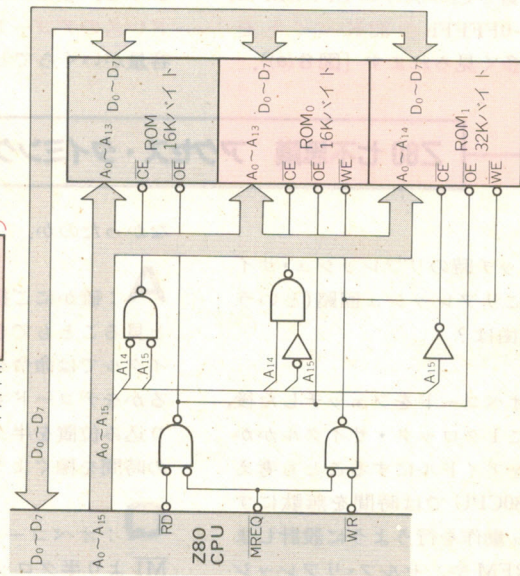
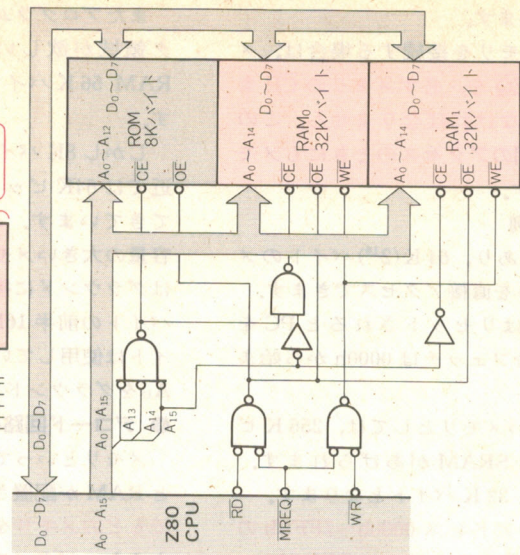
また簡単な動作だけであれば，データの保存にレジスタだけを使い，RAMをいっさい使用しなくてもシステムを構成することも可能ではあります．

しかし扱うデータが多かったり，CALL命令を使ったサブルーチン・コールをするとき，また割り込みを使う場合などは，ワーク・エリアやスタック・エリアが必要となるので，RAMは不可欠です．

● メモリ空間とI/O空間

Z80は8080系の流れをくむCPUですから，メモリ空間とI/O空間の二つのアドレス空間をもっています．このメモリとI/Oのアドレスの指定はアドレス・バスが共用されます．このため，いまCPUがアドレス・バスに出力しているのはメモリのアドレスなのか，I/Oのアドレスなのかを選択するために，$\overline{\text{MREQ}}$と$\overline{\text{IORQ}}$(I/O要求信号)があります．

それぞれ，Lレベルでアクティブになり，当然両方が同時にアクティブになることはありえません．メモリ空間とI/O空間の選択が両方同時に起こることはありえないので，二つの信号を8086のように1本で

〈図8〉ROMとRAMの分割例

(a) ROM32Kバイト，RAM32Kバイトのメモリ・マップ

(b) 32Kバイトのメモリを16Kバイトで使用

(c) ROM16Kバイト，RAM48Kバイトのメモリ・マップ

(d) ROM8Kバイト，RAM56Kバイトのメモリ・マップ

済ませているCPUもあります．

いずれにしろZ80でメモリを接続する場合は，メモリ選択信号である$\overline{MREQ}$を，必ずメモリの$\overline{CE}$または$\overline{OE}$と$\overline{WE}$に接続しなければなりません．この接続を忘れると，I/O空間のアクセスのときにもメモリが選択されてしまいます．

● Z80のメモリ・マップ例

Z80はアドレスが16本あり，64K(2^{16})バイトのメモリ空間(0000～0FFFFh)を直接アクセスできます．

電源投入時など，Z80はリセットされるとPCを0000hにリセットし，命令フェッチは0000hから始まります．

また現在，入手しやすいメモリとしては，256Kビットの容量のEPROMとSRAMがあげられます．256Kビット/8ビットで，32Kバイトとなります．

このことから，前半のアドレス0000h～7FFFhの32Kバイトをプログラム書き込み済みのEPROMに，後半のアドレス8000h～0FFFFhの32KバイトをSRAMにしたシステムが多く見られます［図8(a)］．

またプログラム領域は少なくてよいけれども，データ領域が欲しいという場合はROM 8Kバイト，RAM 56Kバイトといった構成［図8(c)］も可能です．

しかし8Kバイトのメモリが欲しいといっても，最近では64Kビット以下のメモリの入手が困難になってきています．このようなときは，図8(d)のように容量の大きいメモリを使い，余ったアドレス・ビットはグラウンドに接続して使います．この例では，32Kバイトの前半16Kバイトのみを使い，後半の16Kバイトは使用していません．8Kバイトのときは，A_{14}とA_{13}をグラウンドに接続することになります．

● デコード回路のいろいろ

メモリといっても64Kバイトの空間の間に，ROMとRAMが配置されるわけですから，メモリ空間の中でもどのメモリを選択するかの制御が必要です．このような，どのメモリを選択するかの回路をメモリ・アドレスのデコード回路と呼びます．ROMとRAMの容量がいくらでどのアドレス領域に振り分けるかは，

Z80七不思議　アクセス・タイミング編

Q：オペコード・フェッチ時のリフレッシュ・サイクルは不要．Z80CPUにリフレッシュ回路(というかカウンタ)を入れた理由は？

A：Z80CPUでは，オペコードをフェッチした後，命令をデコードするのに1クロック・サイクルかかります．この間，バスをアイドルにすることも考えられますが，これをZ80CPUでは時間を無駄にすることなくリフレッシュ動作を行うように設計しました．今でこそ疑似SRFMや，セルフ・リフレッシュ機能を内蔵し，とくにリフレッシュを意識しなくても良いDRAMがありますが，Z80CPUが開発された頃のマイコンは，リフレッシュを複雑な外付け回路により行っていました．Z80の登場当時は，このリフレッシュ・サイクルは非常に使いかってのよいものでした．ちなみに，このオペコード・フェッチ時にリフレッシュ・サイクルを入れる，というのはザイログ社が特許をもっているアイディアです．

Q：オペコード・フェッチはT_3の立ち上がりで，メモリ・リード/ライト・サイクルはT_3の立ち下がりでデータが取り込まれる．メモリから見ると同じメモリ・リードなのにオペコード・フェッチのほうがアクセス・タイムが厳しいのはなぜか，なぜ同じにしなかったのか．

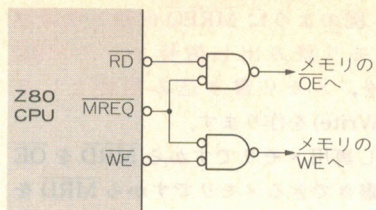
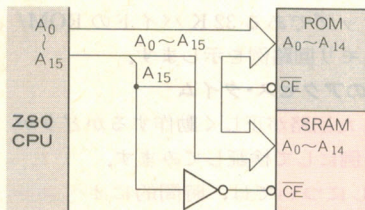
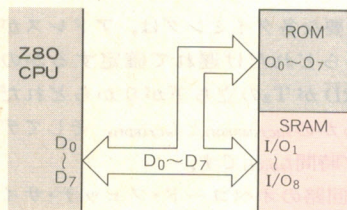
A：確かにご指摘のとおり，同じメモリ・リードと見ることもできますが，オペコード・フェッチ・サイクルでは命令を早く取り込み，その命令が何であるかをデコードする必要があります．そのため，取り込み位置を半クロック早くして，デコードのための時間を稼ぐように設計したのだと思います．

Q：オペコード・フェッチ・サイクル時，なぜ$\overline{M1}$より半クロック遅れて$\overline{MREQ}$が出るのか．$\overline{M1}$が半クロック早く出るなら，オペコード・フェッチ・サイクル時のメモリ・リードをもう半クロック早くして，通常のメモリ・リードと同じアクセス時間にできたのではないか．

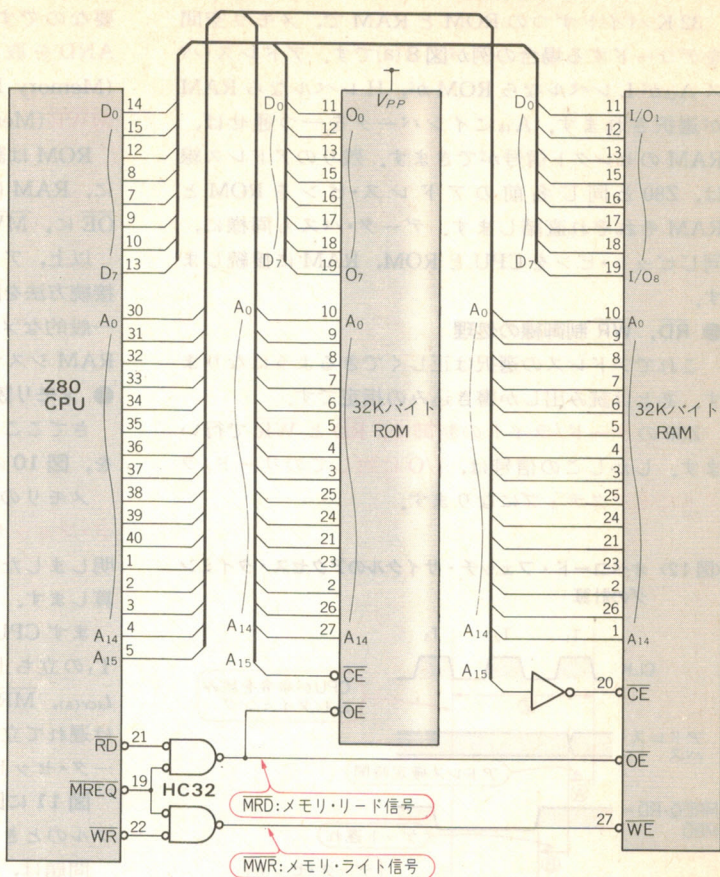
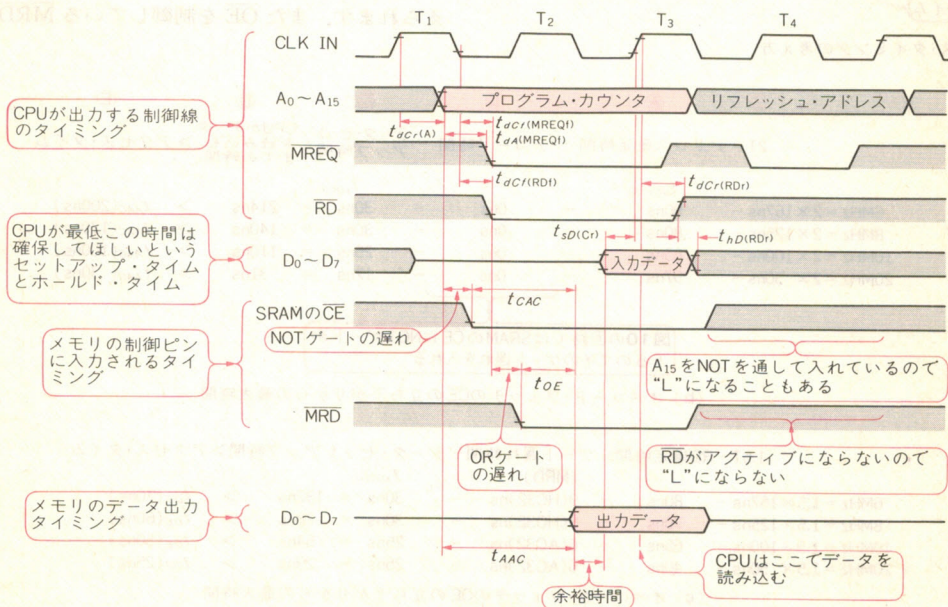
A：これは，あくまでも推測の域を出ませんが，オペコード・フェッチ・サイクル時のRD信号制御を行う回路を，メモリ・リードの回路と共有させるように設計したため，T_1の立ち下がりで変化させるのが一番楽だったからではないかと思います．

また仮に早くしてもアドレス確定時間は変わりませんから，それほどアクセス・タイムを稼ぐことはできないと思われます．

〈信垣育司〉

〈図9〉Z80とメモリとの接続法

(a) $\overline{MREQ}$と$\overline{RD}$, $\overline{WE}$によるメモリ読み書き信号

(b) インバータによるデコード回路 (32KROMと32KROM)

(c) データ・バスの接続

この三つの要素が成り立ってメモリ・アクセスが行われる

〈図10〉32Kバイト ROM/RAMの回路例

〈図11〉オペコード・フェッチ・サイクルの動作

このデコード回路で決めます．

32 K バイトずつの ROM と RAM で，メモリ空間をデコードする場合の例が図 8 (a)です．アドレス・バス A_{15}が L レベルなら ROM が，H レベルなら RAM が選択されます．A_{15}にインバータを一つ通せば，RAM のセレクト信号ができます．残りのアドレス線は，Z80 と同じ名前のアドレス・ピンを ROM と RAM それぞれ直結します．データ・バスも同様に，同じビット・ピンを CPU と ROM，RAM に接続します．

● $\overline{\text{RD}}$，$\overline{\text{WR}}$ 制御線の処理

これでアドレスの選択は正しくできるようになります．あとは読み出しか書き込みの指定です．

Z80 のリード/ライトの制御は，$\overline{\text{RD}}$ と $\overline{\text{WR}}$ で行います．しかしこの信号は，I/O に対してのリード/ライトでもアクティブになります．

〈図 12〉 オペコード・フェッチ・サイクルのアクセス・タイミングの計算

(a) アクセス・タイミングの考え方

(b) オペコード・フェッチの$\overline{\text{CE}}$の立ち下がりからの最大時間

(c) オペコード・フェッチの$\overline{\text{OE}}$の立ち下がりからの最大時間

ここではメモリに対してのリード/ライト信号が必要なのですから，図のように $\overline{\text{MREQ}}$ 信号と負論理 AND を取り，メモリ読み出し信号として $\overline{\text{MRD}}$ (Memory Read)を，メモリ書き込み信号として $\overline{\text{MWR}}$ (Memory Write)を作ります．

ROM は読み出し専用メモリですから $\overline{\text{MRD}}$ を $\overline{\text{OE}}$ に，RAM は読み書きできるメモリですから $\overline{\text{MRD}}$ を $\overline{\text{OE}}$ に，$\overline{\text{MWR}}$ を $\overline{\text{WE}}$ に接続します．

以上，アドレス・バス，データ・バス，各種制御線の接続方法を図 9 にまとめます．また図 10 にもっとも一般的なメモリ・マップである 32 K バイトの ROM/RAM システムのメモリ回路例を示します．

● メモリ読み出しのアクセス・タイム

さてここで設計した回路が正しく動作するかどうかを，図 10 の回路を例にして検証してみます．

メモリの読み出しについては，時間的にオペコード・フェッチ・サイクルがいちばん厳しくなることは説明しました．そこでこの時間でアクセス・タイムを計算します．

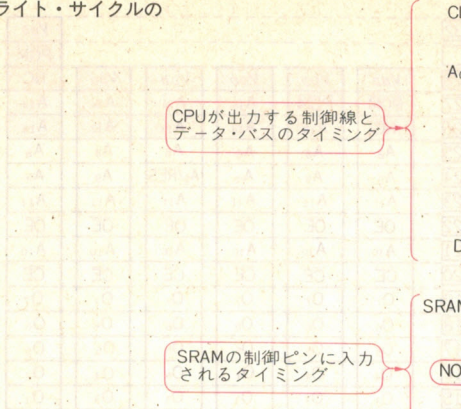
まず CPU 側で重要な各タイミングは，アドレスが T_1の立ち上がりからどれだけ遅れて確定するかの $t_{dCr(A)}$，$\overline{\text{MREQ}}$ や $\overline{\text{RD}}$ が T_2の立ち下がりからどれだけ遅れて立ち下がるかの $t_{dCf(MREQf)}$ と $t_{dCf(RDf)}$，そしてデータ・セット・アップ時間 $t_{sD(Cr)}$ です．

図 11 に図 10 の回路のオペコード・フェッチ・サイクルのときのアクセスのようすを示します．

問題は，デコード回路で使われているロジック IC によるゲート遅れです．ここでは経験的に平均遅延時間の 1.5 倍で計算することにします．

ROM の $\overline{\text{CE}}$ については A_{15}が直接接続されているので，アドレスの確定と $\overline{\text{CE}}$ の立ち下がりは同じと考えられます．また $\overline{\text{OE}}$ を制御している $\overline{\text{MRD}}$ にしても，

〈図 13〉
メモリ・ライト・サイクルの動作

$\overline{\mathrm{MREQ}}$ と $\overline{\mathrm{RD}}$ が立ち下がってからゲートの遅延の時間分遅れてアクティブになります．

また RAM の場合は $\overline{\mathrm{CE}}$ に A_{15}にインバータを通した信号を接続しているので，$\overline{\mathrm{CE}}$ がアドレスの確定より若干遅れます．この遅れは考慮にいれなければなりません．

これらオペコード・フェッチ・サイクルのアクセス・タイムの計算例を図 12 に示します．これを満足できれば，メモリ・リード・サイクルに対しても十分アクセス・タイムに余裕ができる計算になります．

● $\overline{\mathrm{MREQ}}$ を $\overline{\mathrm{CE}}$ に入れるか，$\overline{\mathrm{OE}}$ や $\overline{\mathrm{WE}}$ に入れるか

図 8 や図 10 の回路例では $\overline{\mathrm{MREQ}}$ を $\overline{\mathrm{RD}}$ や $\overline{\mathrm{WR}}$ と負論理 AND をとってメモリの $\overline{\mathrm{OE}}$ や $\overline{\mathrm{WE}}$ に接続しています．Z80 ではメモリ空間と I/O 空間の区別があり，どちらの空間に対するアクセスなのかを示す制御線として $\overline{\mathrm{MREQ}}$ や $\overline{\mathrm{IORQ}}$ があるのはすでに説明したとおりです．

メモリか I/O かの選択ですから，どちらかといえばアドレス・デコーダ回路に接続して $\overline{\mathrm{CE}}$ で制御すべきところです．しかし Z80 の場合，アドレスが T_1の立ち上がりで確定するのに対し，$\overline{\mathrm{MREQ}}$ は T_1の立ち下がりでアクティブになるので，半クロックだけ遅れることになります．これをこのまま $\overline{\mathrm{CE}}$ に接続すると，$\overline{\mathrm{CE}}$ がイネーブルになってからのアクセス時間 t_{CAC}時間だけさらに遅れることになり，メモリ・アクセス・サイクルがさらに厳しくなります．

このため Z80 においては，$\overline{\mathrm{MREQ}}$ を読み書き信号である $\overline{\mathrm{RD}}$ や $\overline{\mathrm{WR}}$ と負論理 AND を取り，アクセス時間を稼ぎます．

ただし後述のバックアップ回路などの例のように，例外的な場合もあります．

● **メモリ書き込みのアクセス・タイム**

次にメモリ・ライト・サイクルです．ほとんどの SRAM は $\overline{\mathrm{WR}}$ の立ち下がり時にアドレスが確定していればよいので，CPU のアドレスの遅延はほとんど考えなくてもだいじょうぶでしょう．注目すべき点は，$\overline{\mathrm{WR}}$ の立ち上がりに対するデータ・セットアップ時間と，$\overline{\mathrm{WR}}$ クロックのパルス幅です．

図 10 の回路のメモリ・ライト・サイクルにおけるアクセス・タイムのようすを図 13 に示します．CPU の $\overline{\mathrm{WR}}$ の立ち下がりに対するデータ・バスのデータ確定時間 $t_{dD(WRf)}$ と，$\overline{\mathrm{WR}}$ パルスの時間を加算すれば，$\overline{\mathrm{WR}}$ の立ち上がり対するデータ・バスのデータ確定時間が求められます．

これが SRAM の書き込みタイミングである，$\overline{\mathrm{MWR}}$ の立ち上がりに対してのデータ・セットアップ時間より十分長いかどうかを比較します．

また $\overline{\mathrm{WR}}$ クロックのパルス幅については，CPU の $\overline{\mathrm{WR}}$ と SRAM の $\overline{\mathrm{WE}}$ の間に $\overline{\mathrm{MREQ}}$ との負論理 AND ゲートがありますが，L レベルから H レベル，H レベルから L レベルの遅延は HC タイプや AC タイプではほぼ同じと考えると，CPU の最小 $\overline{\mathrm{WR}}$ パルス幅である t_{wWR}と同じ時間が SRAM の $\overline{\mathrm{WE}}$ に入力されていると考えられます．

またこうしてアクセス・タイムを計算しても，MOS タイプのメモリなどは DRAM に限らず，高温でアクセス・タイムが遅くなるので，低温で動作しても安心はできません．

〈図14〉 64 K～4 M ビットの各種メモリのピン配置

32ピン	28ピン	1M	512K	256K フラッシュ	256K	128K	64K
1		V_{PP}					
2		A_{16}					
3	1	A_{15}	A_{15}	V_{PP}	V_{PP}	V_{PP}	V_{PP}
4	2	A_{12}	A_{12}	A_{12}	A_{12}	A_{12}	A_{12}
5	3	A_7	A_7	A_7	A_7	A_7	A_7
6	4	A_6	A_6	A_6	A_6	A_6	A_6
7	5	A_5	A_5	A_5	A_5	A_5	A_5
8	6	A_4	A_4	A_4	A_4	A_4	A_4
9	7	A_3	A_3	A_3	A_3	A_3	A_3
10	8	A_2	A_2	A_2	A_2	A_2	A_2
11	9	A_1	A_1	A_1	A_1	A_1	A_1
12	10	A_0	A_0	A_0	A_0	A_0	A_0
13	11	O_0	O_0	O_0	O_0	O_0	O_0
14	12	O_1	O_1	O_1	O_1	O_1	O_1
15	13	O_2	O_2	O_2	O_2	O_2	O_2
16	14	GND	GND	GND	GND	GND	GND

32ピン	28ピン	64K	128K	256K	256K フラッシュ	512K	1M	4M
32							V_{DD}	
31							$\overline{PGM}$	A_{18}
30	28	V_{DD}	V_{DD}	V_{DD}	V_{DD}	V_{DD}	NC	A_{17}
29	27	$\overline{PGM}$	$\overline{PGM}$	A_{14}	A_{14}	A_{14}	A_{14}	
28	26	NC	A_{13}	A_{13}	A_{13}	A_{13}	A_{13}	
27	25	A_8	A_8	A_8	A_8	A_8	A_8	
26	24	A_9	A_9	A_9	A_9/RES	A_9	A_9	
25	23	A_{11}	A_{11}	A_{11}	A_{11}	A_{11}	A_{11}	
24	22	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	
23	21	A_{10}	A_{10}	A_{10}	A_{10}	A_{10}	A_{10}	
22	20	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	$\overline{CE}$	
21	19	O_7	O_7	O_7	O_7	O_7	O_7	
20	18	O_6	O_6	O_6	O_6	O_6	O_6	
19	17	O_5	O_5	O_5	O_5	O_5	O_5	
18	16	O_4	O_4	O_4	O_4	O_4	O_4	
17	15	O_3	O_3	O_3	O_3	O_3	O_3	

2M & 4M

(a)　UVEPROMとフラッシュROMのピン配置

32ピン	28ピン	4M SRAM	1M SRAM	256K EEPROM	256K SRAM	64K SRAM
1		A_{18}	NC			
2		A_{16}	A_{16}			
3	1	A_{14}	A_{14}	A_{14}	A_{14}	NC
4	2	A_{12}	A_{12}	A_{12}	A_{12}	A_{12}
5	3	A_7	A_7	A_7	A_7	A_7
6	4	A_6	A_6	A_6	A_6	A_6
7	5	A_5	A_5	A_5	A_5	A_5
8	6	A_4	A_4	A_4	A_4	A_4
9	7	A_3	A_3	A_3	A_3	A_3
10	8	A_2	A_2	A_2	A_2	A_2
11	9	A_1	A_1	A_1	A_1	A_1
12	10	A_0	A_0	A_0	A_0	A_0
13	11	I/O_0	I/O_0	I/O_0	I/O_0	I/O_0
14	12	I/O_1	I/O_1	I/O_1	I/O_1	I/O_1
15	13	I/O_2	I/O_2	I/O_2	I/O_2	I/O_2
16	14	GND	GND	GND	GND	GND

32ピン	28ピン	64K SRAM	256K SRAM	256K EEPROM	1M SRAM	4M SRAM
32					V_{DD}	V_{DD}
31					A_{15}	A_{15}
30	28	V_{DD}	V_{DD}	V_{DD}	CS_2	A_{17}
29	27	$\overline{WE}$	$\overline{WE}$	$\overline{WE}$	$\overline{WE}$	$\overline{WE}$
28	26	CS	A_{13}	A_{13}	A_{13}	A_{13}
27	25	A_8	A_8	A_8	A_8	A_8
26	24	A_9	A_9	A_9	A_9	A_9
25	23	A_{11}	A_{11}	A_{11}	A_{11}	A_{11}
24	22	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$	$\overline{OE}$
23	21	A_{10}	A_{10}	A_{10}	A_{10}	A_{10}
22	20	$\overline{CS}$	$\overline{CS}$	$\overline{CS}$	$\overline{CS}$	$\overline{CS}$
21	19	I/O_7	I/O_7	I/O_7	I/O_7	I/O_7
20	18	I/O_6	I/O_6	I/O_6	I/O_6	I/O_6
19	17	I/O_5	I/O_5	I/O_5	I/O_5	I/O_5
18	16	I/O_4	I/O_4	I/O_4	I/O_4	I/O_4
17	15	I/O_3	I/O_3	I/O_3	I/O_3	I/O_3

(b)　SRAMとEEPROMのピン配置

当然，さらに複雑なデコード回路や，後述するバックアップ回路のためのゲートを追加すれば，よりアクセス速度の速いメモリが必要になります．

● 64 K～256 K ビット・メモリの対応

図14に64 K～4 M ビットのEPROMとSRAMのピン配置図を示します．

256 K ビットのメモリの場合，1番ピンと27番ピン以外のピンは，ROMもRAMも同じピン配置である点を頭に入れておくとよいでしょう．SRAMとEEPROMは，1番ピンがA_{14}，27番ピンが$\overline{WE}$信号です．UV EPROMとフラッシュROMは，1番ピンがV_{PP}，27番ピンがA_{14}です．

このフラッシュROMはUV EPROMとピン配置がまったく同じなので，パターンの変更なしで置き換えができます．EEPROMを使う場合は，あらかじめ切り替え回路をパターン化しておきます．

また64K～256 K ビットのメモリに対応するためのジャンパ・ピンの回路を図15に示します．これで同じ28ピン・パッケージであることを利用して，ジャンパ・ピンだけで異なる容量のメモリにも対応させることができます．

またこのピン配置を考えると，64 K ビットや128 K ビット用のROMソケットに256 K ビットのEPROMを差し込むこともできます．ただし，A_{14}やA_{13}にあたるピンの処理で注意する点があります．

これらのピンの基板上の処理によっては，ROMライタでの書き込み開始アドレスが，ROMの物理的な0000hからでなく，6000hや4000hといったA_{14}やA_{13}

〈図 15〉 64 K～256 K ビット ROM と RAM 対応ジャンパ回路

（a） 64K～256KビットEPROM対応のジャンパ回路1

（b） 64K～256KビットEPROM対応のジャンパ回路2

（c） 64K, 256KビットSRAM対応のジャンパ回路

〈図 16〉
バンク切り替えの例

アドレス　メモリ64Kバイト

0000h　プログラムROM　ROM 32Kバイト
7FFFh
8000h　裏RAM 0 1 2 3 4 5 6 7　この領域をバンク切り替えする
BFFFh
C000h　ワーク・エリア　スタック・エリア　表RAM 32Kバイト
FFFFh

（a） バンク切り替えのメモリ・マップ図

I/Oポートに設けたバンク切り替え用レジスタの下位4ビット

ポートA				選択されるメモリ
PA3	PA2	PA1	PA0	
0	×	×	×	表RAM
1	0	0	0	裏RAM0
1	0	0	1	1
1	0	1	0	2
1	0	1	1	3
1	1	0	0	4
1	1	0	1	5
1	1	1	0	6
1	1	1	1	7

×：don't care

（b） バンク・セレクト・ビット表

アドレス・バス　A0～A13　A14　A15
表RAM/裏RAM セレクト・ビット
パラレルI/Oポートから（第4章参照）　PA3　PA2　PA1　PA0
表RAM 32Kバイト　A0～A13　A14　CE
裏RAM 128Kバイト　CE1　CE2　A16　A15　A14　A13～A0
バンク・セレクト・ビット

（c） バンク切り替え回路

のビットが"1"であるアドレスから書き込まなければならない点です．CPU側はアドレス0000hからプログラムを読み込もうとしますが，ROMにしてみれば，A_{14}やA_{13}のビットが"1"であるアドレスに対してアクセスされることになるからです．

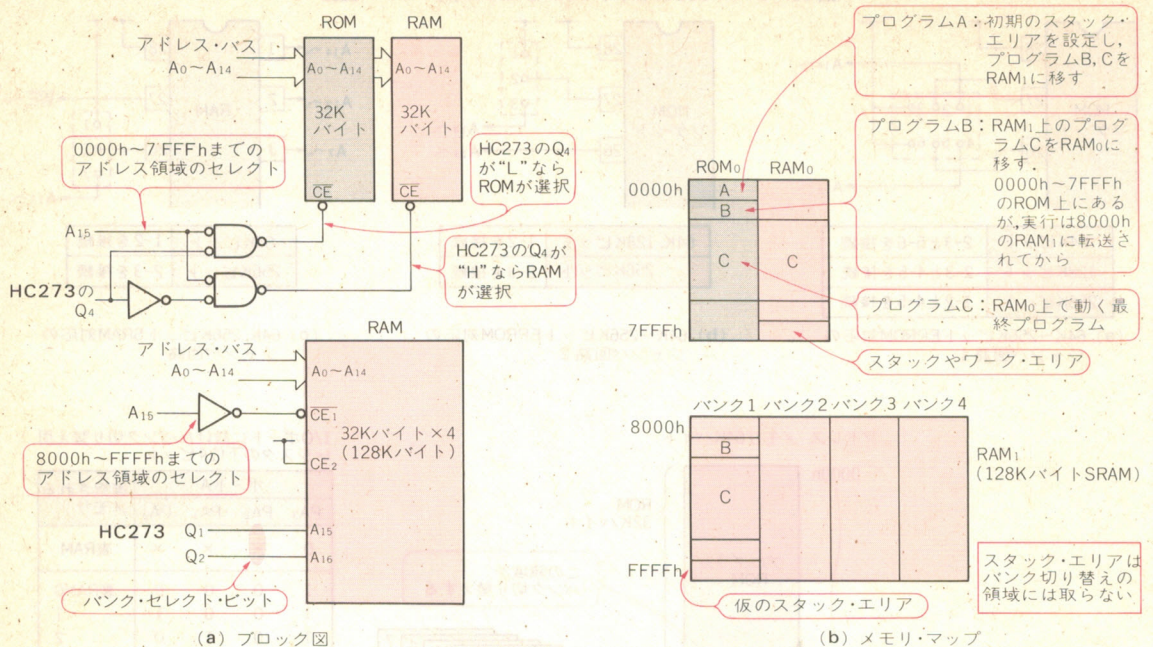
またこのアドレスは，基板の回路が図15(a)のようになっているか，図15(b)のようになっているかでも違います．図(a)ではA_{13}はHレベル固定ですが，図(b)では CPU のA_{13}が接続されることになります．

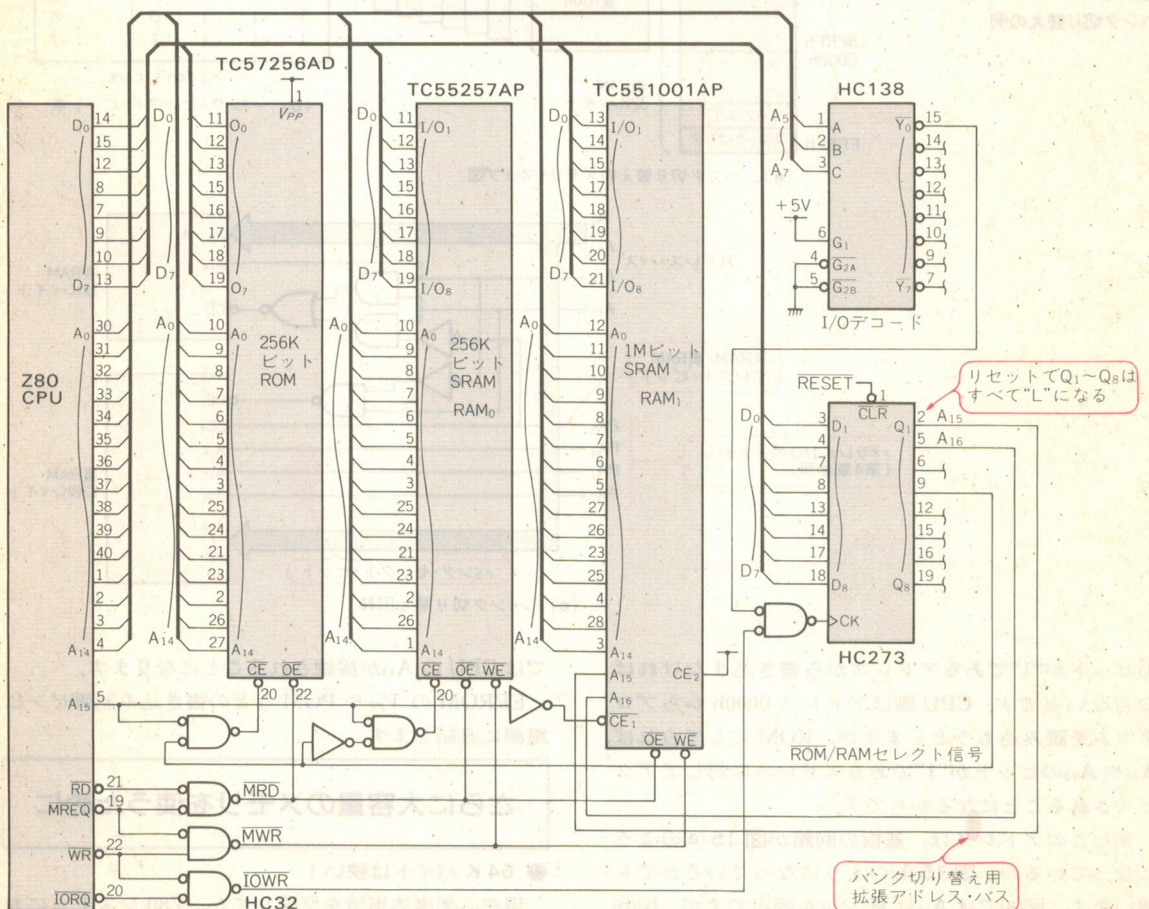
EPROM の V_{PP}や $\overline{PGM}$ などの書き込み制御ピンは，電源に直結します．

さらに大容量のメモリを使うために

● 64 K バイトは狭い！

現在の半導体事情を反映してか，Z80 システムにお

〈図 17〉 64 K バイト・オールRAM 領域にできるメモリ回路

〈図 18〉 バンク切り替えシステムの具体的な回路例

Z80 CPU
TC57256AD 256K ビット ROM
TC55257AP 256K ビット SRAM RAM0
TC551001AP 1M ビット SRAM RAM1
HC138 I/Oデコード
HC273
HC32
MRD
MWR
IOWR
RESET
リセットでQ1～Q8はすべて"L"になる
ROM/RAMセレクト信号
バンク切り替え用拡張アドレス・バス

いても64Kバイトのメモリ容量に限界が見えてきているのも事実です．最近はC言語などの高級言語でプログラムを組むことが多く，少しでも凝ったことをやり始めると，64Kバイトではあっという間にメモリを使いきってしまうことがあります．

しかし，Z80には16本のアドレス・バスしかありません．16本ではどう指を折って数えても64Kバイト以上のメモリを数えられません．Z80で64Kバイト以上のメモリを使う方法はないのでしょうか．

そこでどこからか17，18本目のアドレス・バスを無理矢理作り，64Kバイト以上のRAMを管理したり，64Kバイト以内のあるアドレス領域を切り替えて使い，トータル的に64Kバイト以上のメモリを使うようなシステムが考え出されました．このようなシステムをバンク切り替えと呼びます．

● メモリ・バンク切り替えの具体的例

バンク切り替えでは，ワーク・エリアとスタック・エリアを切り替わらないアドレス領域に確保しないと，CPUは暴走(プログラムの動作がおかしくなる)してしまいます．また切り替わるアドレス領域にPCがあるときにバンク切り替えしても暴走します．それはそうです．切り替えたとたんプログラム自身がメモリ・バンクの裏へ消えてしまうのですから．

図16にアドレス8000h～0BFFFhにバンク切り替え用の窓を開け，バンク切り替え領域の大きさを16Kバイトとした例を示します．

I/O空間に設けたPIO(パラレルI/O)についての設計法は，次章を参照していただくとして，とりあずここでは単にPIOのポートAの下位4ビットをバンク選択用に使うとだけ説明しておきます．またリセット状態でPA_0～PA_3はLレベルになります．

PA_3がLレベルなら表RAMである256KビットSRAMが，CPUに対してアドレス8000h～0FFFFhに選択されます．Hレベルならアドレス8000h～0BFFFhの領域が裏RAMになります．

さらに裏RAMには1MビットSRAMが使われているので，1M/8ビットで128Kバイトの容量があります．これを16Kバイトのバンクに分けるので，全部で8バンク切り替えが可能になります．この8バンクをPA_2～PA_0の3ビットで選択します．

バンクの切り替えはOUT命令で，選択したいバンクの値をPIOのポートAの下位4ビットに出力して設定します．

● 64Kバイト・オールRAM

システムによっては，ROM領域までRAMにしたいときがあります．アドレスの前半の32Kバイトの表にROM，裏にRAM，後半の32KバイトにRAMのバンクを四つ配置したシステム例を図17に示します．

〈リスト 1〉バンク切り替えプログラム例

```
;**************************************************************
;*  Z80 BANK TEST PROGRAM    at Mar.20.1994  by H.MURATA    *
;**************************************************************
; *** MEMORY SYMBOL ***
STACK0  EQU     0000H   ;SP=FFFF+1 START
STACK1  EQU     8000H   ;SP=7FFF+1 RESTART
; *** I/O    SYMBOL ***
IOCE01  EQU     000H    ;LOW MEMORY SELECT

;**************************************************************
;*              PROGRAM    START                             *
;**************************************************************
        ORG     0000H
        LD      SP,STACK0

        LD      HL,0100H            ;RAM0(アドレス 0100h)から
        LD      DE,8100H            ;RAM1(アドレス 8100h)へ
        LD      BC,7700H            ;7700hバイト
        LDIR                        ;転送
        JP      8100H               ;転送したプログラムへジャンプ
;
        ORG     0100H               ;8100hへ転送されて実行される
        LD      A,008H              ;HC273 Q4 '1'
        OUT     (IOCE0),A           ;バンク切り替え
;
        LD      HL,8200H            ;RAM1(アドレス 8200h)から
        LD      DE,0200H            ;RAM2(アドレス 0200h)へ
        LD      BC,7600H            ;7600hバイト
        LDIR                        ;転送
        JP      0200H               ;転送したプログラムへジャンプ
;
        ORG     0200H               ;リスタート RAM0
        LD      SP,STACK1
        ;
        ;                 以下、プログラムが続く
        ;
        END
```

リセット後はHC273のQ_1～Q_4がLレベルになるので，アドレス0000h～7FFFhにはROMが選択されます．Z80はリセット後，アドレス0000hから実行を開始するので，必ずROMが選択されなくてはなりません．よってHC273などのリセット機能を備えたラッチを使用します．

そしてプログラムAにより仮のスタック・エリアを設定し，BとCの領域をブロック転送命令でアドレス8000h以降のRAM領域に転送します．

次にプログラムBでバンク・セレクト・ビットのQ_4をHレベルにし，アドレス0000h～7FFFhの領域にRAM_0を選択します．そして先ほどとは逆方向に，アドレス8000hからのデータをアドレス0000hの領域に転送します．これで64KバイトすべてRAMのシステムができます．

またこの回路では，アドレス8000h以降の領域もバンク切り替え可能です．RAM_1～RAM_4の切り替えは，HC273のQ_1，Q_2で選択します．

当然ながら，この領域にスタック・エリアを確保している場合は，アドレス0000h～7FFFhの領域に再確保しないと，バンク切り替えしたとたんにスタック領域がバンクの裏に消えてアクセスできなくなってしまいます．

またこの図17の回路を具体的な回路図にしたのが図18で，この回路のコントロール・プログラム例を

〈図19〉ライト・プロテクト回路の例

(a) ライト・プロテクト回路(RAM全体のプロテクト)

(b) 特定のアドレス領域だけをプロテクト

〈図20〉スタンバイ回路

TMPZ84C61APなどのCGC(クロック・ジェネレータ・コントローラ)などを使うことにより,スタンバイ機能をもたせることができる.またZ84C015などに内蔵されているCGCでも同様

(a) $\overline{MREQ}$を$\overline{CE}$へ接続

HALT命令実行後は$\overline{HALT}$が"L"になるので,ROMとRAMの双方の$\overline{CE}$を"H"にできる.

A15

HALT

ROMのCEへ

RAMのCEへ

RD

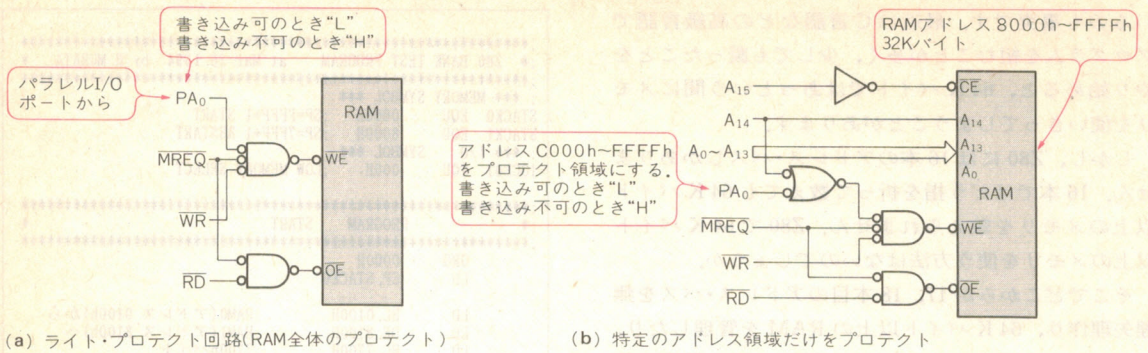
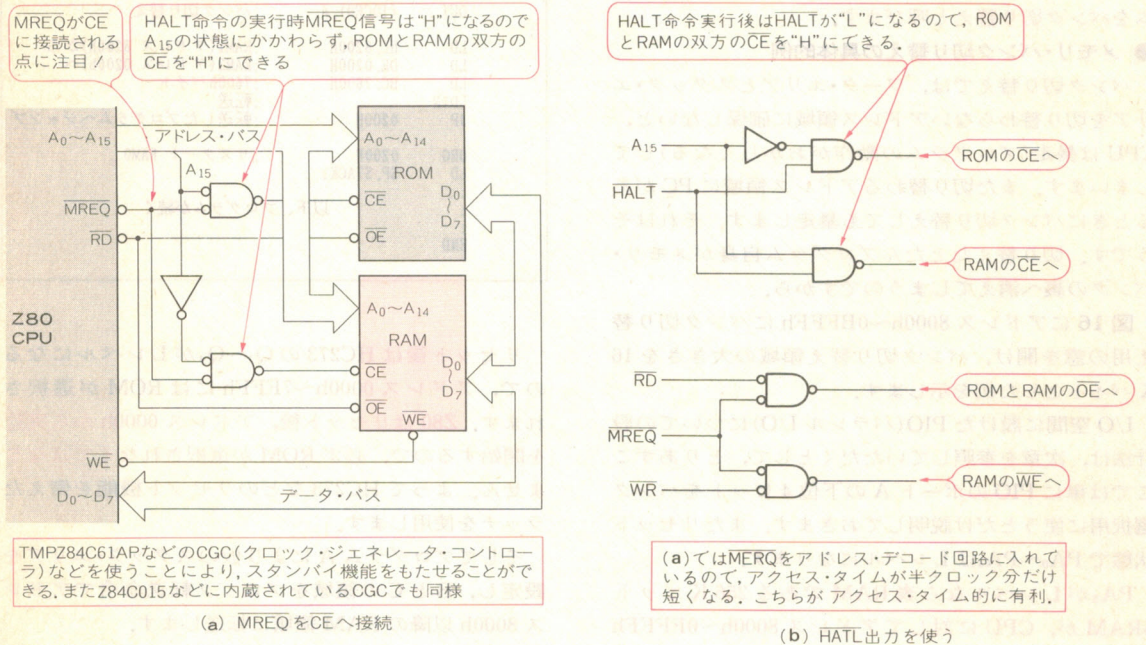
ROMとRAMのOEへ

MREQ

WR

RAMのWEへ

(a)では$\overline{MERQ}$をアドレス・デコード回路に入れているので,アクセス・タイムが半クロック分だけ短くなる.こちらがアクセス・タイム的に有利.

(b) $\overline{HATL}$出力を使う

リスト1に示します.

● **RAMのプロテクト回路例**

通常は普通に読み書きができ,ある特定の状態のときに書き込み禁止(プロテクト)をかけられる回路の例を図19に示します.

これもバンク切り替え回路と同じように,任意のI/O空間にパラレル・ポートをつけて,そのポートの任意のビットを書き込み可/不可フラグとして使用します.

この信号と$\overline{WR}$信号(実際には$\overline{MREQ}$と$\overline{WR}$の負論理ANDを取った$\overline{MWR}$)を負論理ANDをとってRAMの書き込み信号にすればよいわけです.

図19の例では,ポートのビットをHレベルにすると書き込み禁止となります.

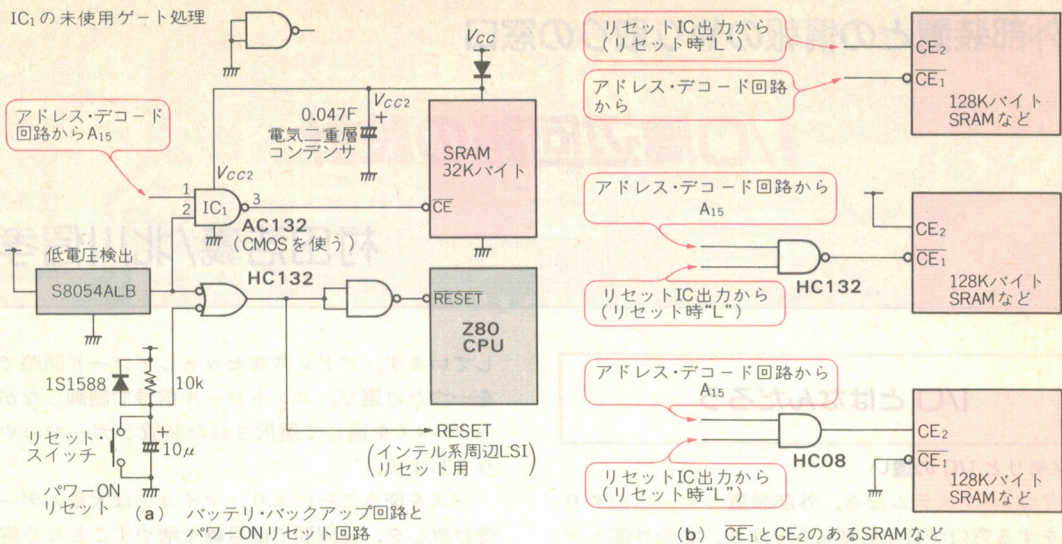
また場合によっては,任意のアドレス領域にだけプロテクトをかけたいときもあるでしょう.このときは図19(b)のように,プロテクトをかけたいアドレス領域をデコードしてプロテクト回路に入力すればよいわけです.

アドレスのデコード回路を変更すれば,任意のアドレス領域のプロテクトが実現できます.もっとも,任意といえどもROM領域はもともと書き込みできないので意味はありませんが.

また,このプロテクト回路についてだけではありませんが,CPUからのコントロール信号とメモリとの間にゲートが入ると,それだけゲート遅延時間が増加することになります.その分だけ高速なアクセス・タイムのメモリを使うか,遅延時間の少ないゲートを使

〈図21〉RAMのバッテリ・バックアップ回路

(a) バッテリ・バックアップ回路とパワーONリセット回路

(b) $\overline{CE_1}$とCE₂のあるSRAMなど

うなどの工夫が必要です．

低消費電力モードとバックアップ

● バッテリで動作させる

Z80もCMOS化されて省電力になり，マイコン・システム全体をバッテリ動作させることができるようになりました．専用のクロック・ジェネレータを使ってシステム全体にスタンバイ機能をもたせることにより，バッテリの寿命を延ばすことができます．

低消費電力対応のメモリは，$\overline{CE}$がHレベルであれば非選択状態であると認識し，低消費電力モードに入ります．

そこで図20のように，メモリのデコード回路に$\overline{MREQ}$信号を接続し，$\overline{RD}$，$\overline{WR}$信号はメモリの$\overline{OE}$や$\overline{WE}$に直接接続します．スタンバイ時(HALT命令を実行したとき)，$\overline{MREQ}$はHレベルになるので，メモリをスタンバイ状態にすることができます．

CPUをHALT命令で停止させた状態からもう一度プログラムを実行させるには割り込みを使いますが，ここでは説明しません．

ただしすでに説明したように$\overline{CE}$に$\overline{MREQ}$信号を入れるので，アクセス・タイムにはこのことを十分考慮に入れて設計してください．

図20(b)は，$\overline{MREQ}$ではなく$\overline{HALT}$を使った例です．$\overline{MREQ}$は通常どおり$\overline{RD}$や$\overline{WR}$と接続しているので，アクセス時間的には$\overline{HALT}$を$\overline{CE}$に接続するゲート分だけ気にすればよいので，(a)の回路よりはアクセス・タイムに余裕があります．

またROMのアクセス・タイムを稼ぐために，$\overline{CE}$をグラウンドに接続しっぱなしにする方法は，当然のことながら，この低消費電力モードにはならなくなります．

● SRAMのバッテリ・バックアップ

継続的なデータ収拾システムや，RAM領域に製作途中のプログラムをいれてデバッグする開発中のマイコン・システムでは，電源が瞬断した場合などに備えてSRAMの回路に，バッテリ・バックアップ回路を付加すると便利です．

またたいていのCPUは，電源の瞬断が起こると暴走し，RAMに誤書き込みしてしまいます．この誤書き込みも防止することができます．

このためには電源検出用に便利な，専用のリセットICがあります．

基本的には，バックアップ時にSRAMの$\overline{CE}$端子をHレベル(2V以上)になるようにし，別電源(電池やコンデンサ)に切り替えます．

図21に，電圧低下の検出用にS805シリーズを使った例を示します．パワーONリセット回路と負論理のORを取り，CPUにリセットをかけます．

この回路のポイントは，SRAMの$\overline{CE}$につながったAC132の電源を電気二重層コンデンサから取り，SRAMとAC132の両方をバックアップしている点です．AC132も動作させていることで，SRAMの$\overline{CE}$はHレベルに保たれるからです．

◆参考文献◆

(1) 鈴木八十二；半導体MOSメモリとその使い方，日刊工業新聞社．
(2) ㈱東芝，MOSメモリデータブックROM編，1991年．
(3) ㈱東芝，MOSメモリデータブックRAM編，1993年．

(トランジスタ技術1994年6月号に加筆，修正)

第4章

外部装置との情報のやり取りの窓口

I/O周辺回路の設計

村田浩義/北川信孝

I/O とはなんだろう

● **メモリと I/O の違い**

マイコン・システムなら，外部装置とデータのやり取りをする窓口が必ず必要になります．この外部とデータをやり取りする窓口にあたる部分をI/Oと呼びます(図1)．メモリもデータ・バスに直結されますが，外部装置とデータのやりとりはしないのでI/Oとはいいません．

第1章でも説明したように，CPUに接続するメモリとI/Oの接続回路の違いは，おおざっぱにいうと，基本的には $\overline{\mathrm{MREQ}}$ がアクティブになるか $\overline{\mathrm{IORQ}}$ がアクティブになるかの違いだけです．アドレス・バスが出力され，$\overline{\mathrm{RD}}$ または $\overline{\mathrm{WR}}$ により読み出しか書き込みの選択をする部分は，メモリについてもI/Oについても基本的に同じです．

しかしソフトウェア的な立場から見たときには，非常に大きな違いがあります．

● **I/O デバイスとは**

マイコン・システムでは，各種バスが重要な働きをしています．アドレスをセットしデコード回路で相手を一つだけ選び，コントロール信号で制御しながらデータ・バスを通じて選択された装置とデータのやりとりをします．

バスを使うことにより，マイコンは大量のデータの受け渡しを，装置間で信号線を増やすことなく容易に行えます．

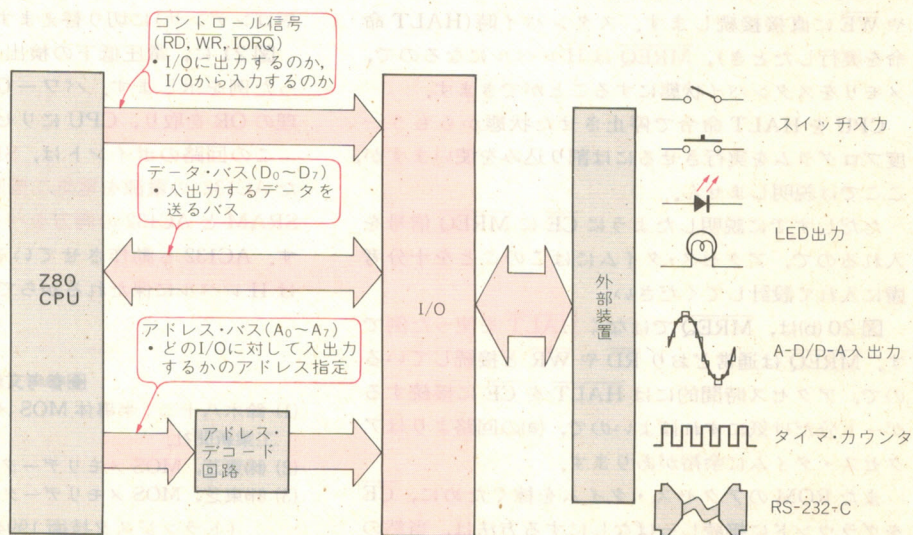
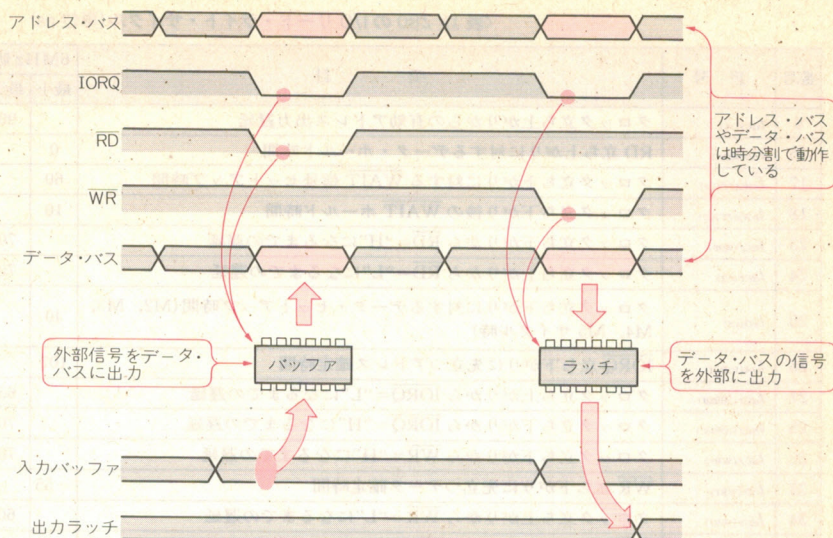
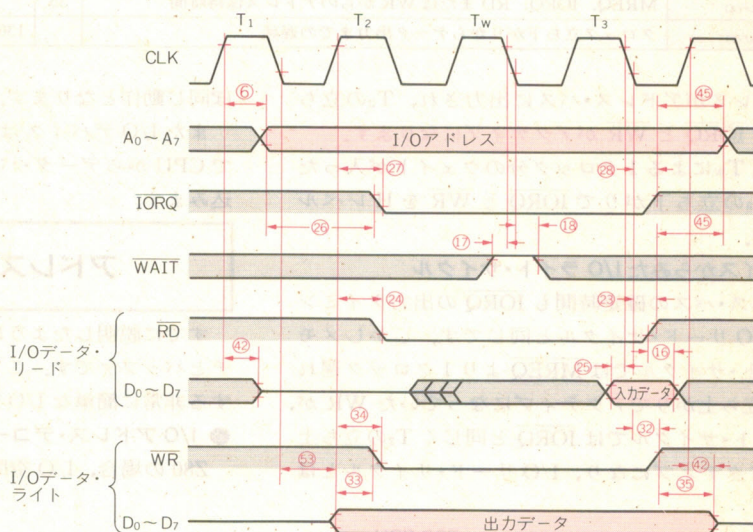
この反面，バスは時分割でしか使えませんから，外部装置側にデータをラッチする機能とバスをバッファリングする機能が必要になります(図2)．

このように，CPUが外部のデータを読み込みたいときに，外部からのデータをデータ・バスに出力するもの，またCPUが外部にデータを出力したいときに，データ・バスからデータを入力して外部に出力するものなど，入出力をコントロールするラッチやバッファがI/Oデバイスなのです．

Z80 の I/O アクセス・タイミング

● **I/O リード・サイクル**

I/Oに対するアクセスは，I/O制御命令であるIN/

〈図1〉
I/Oとは

〈図2〉
I/Oデバイスとバスの関係

〈図3〉
Z80のI/Oリード・ライト・サイクルのタイミング

OUT/OTIR/INIRなどの命令が実行されたときに発生します．

Z80のI/Oリード・ライト・サイクルを図3に，アクセス・タイミングを表1に示します．Z80のI/Oリード/ライト・サイクルは，図中のT_Wで示すウェイト・サイクルが，CPU自身によって強制的に1クロック挿入されるので，4クロック・サイクルで実行されます．

IN命令の実行，つまりI/Oリード・サイクルでは，まずT_1の立ち上がりでI/Oポート・アドレス(A_0～A_7)がアドレス・バスに出力され，T_2の立ち上がりで$\overline{IORQ}$と$\overline{RD}$がアクティブになります．

そしてT_Wによる1クロック分のウェイトが入ったあと，T_3の立ち下がりでCPUはI/Oポートから出力されたデータを取り込みます．

● **デバイスからみたI/Oリード・サイクル**

メモリ・リード・サイクルではT_1の立ち下がりで出力されていた$\overline{MREQ}$や$\overline{RD}$が，I/Oリード・サイクルでは半クロック遅くなり，T_2の立ち上がりで出力されます．しかしT_Wによるウェイトが1クロック入るので，実質的にはT_2とT_WそしてT_3の立ち下がりまでと，アドレスが確定してから3.5クロック，$\overline{IORQ}$がアクティブになってから2.5クロック分の時間があり，メモリ・リード/ライト・サイクルよりアクセス時間に余裕があります．

● **I/Oライト・サイクル**

OUT命令の実行，つまりI/Oライト・サイクルでは，I/Oリード・サイクルと同様にT_1の立ち上がりで

〈表1〉Z80のI/Oリード・ライト・サイクル(単位ns)

番号	記号	項目	6MHz版 最小	6MHz版 最大	8MHz版 最小	8MHz版 最大	10MHz版 最小	10MHz版 最大	20MHz版 最小	20MHz版 最大
6	$t_{dCr(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
16	$t_{hD(RDr)}$	RD立ち上がりに対するデータ・ホールド時間	0		0		0		0	
17	$t_{sWAIT(Cf)}$	クロック立ち下がりに対する$\overline{WAIT}$信号セットアップ時間	60		50		20		7.5	
18	$t_{hWAIT(Cf)}$	クロック立ち下がり後の$\overline{WAIT}$ホールド時間	10		10		10		10	
23	$t_{dCf(RDr)}$	クロック立ち下がりから$\overline{RD}$="H"になるまでの遅延		70		60		55		40
24	$t_{dCr(Cf)}$	クロック立ち上がりから$\overline{RD}$="L"になるまでの遅延		70		60		55		40
25	$t_{sD(Cf)}$	クロック立ち下がりに対するデータ・セットアップ時間(M2, M3, M4, M5サイクル時)	40		30		25		12	
26	$t_{dA(IORQf)}$	$\overline{IORQ}$立ち下がりに先立つアドレス確定時間	107		75		50		0	
27	$t_{dCr(IORQf)}$	クロック立ち上がりから$\overline{IORQ}$="L"になるまでの遅延		65		55		50		40
28	$t_{dCf(IORQr)}$	クロック立ち下がりから$\overline{IORQ}$="H"になるまでの遅延		70		60		55		40
32	$t_{dCf(WRr)}$	クロック立ち下がりから$\overline{WR}$="H"になるまでの遅延		70		60		55		40
33	$t_{dD(WRf)}$	$\overline{WR}$立ち下がりに先立つデータ確定時間	−55		−55		−10		−30	
34	$t_{dCr(WRf)}$	クロック立ち上がりから$\overline{WR}$="L"になるまでの遅延		60		60		50		40
35	$t_{dWRr(D)}$	$\overline{WR}$="H"になってからの出力データ保持時間	30		15		10		0	
42	$t_{dCr(Dz)}$	クロック立ち上がりからデータ・バス・フロート状態までの遅延		80		70		65		40
45	$t_{dCr(A)}$	$\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$または$\overline{WR}$からのアドレス保持時間	35		20		20		0	
53	$t_{dCf(D)}$	クロック立ち下がりからデータ出力までの遅延		130		115		110		75

I/Oアドレスがアドレス・バスに出力され，T_2の立ち上がりで$\overline{IORQ}$と$\overline{WR}$がアクティブになります．

そしてT_Wによる1クロック分のウェイトが入ったあと，T_3の立ち下がりで$\overline{IORQ}$と$\overline{WR}$をHレベルにします．

● デバイスからみたI/Oライト・サイクル

アドレス・バスの確定時間も$\overline{IORQ}$の出力タイミングも，I/Oリード・サイクルと同じです．しかしメモリ・ライト・サイクルでは$\overline{MREQ}$より1クロック遅れてT_2の立ち上がりでアクティブになっていた$\overline{WR}$が，I/Oライト・サイクルでは$\overline{IORQ}$と同じくT_2の立ち上がりでアクティブになり，I/Oリード・サイクルとほぼ同じ動作となります．

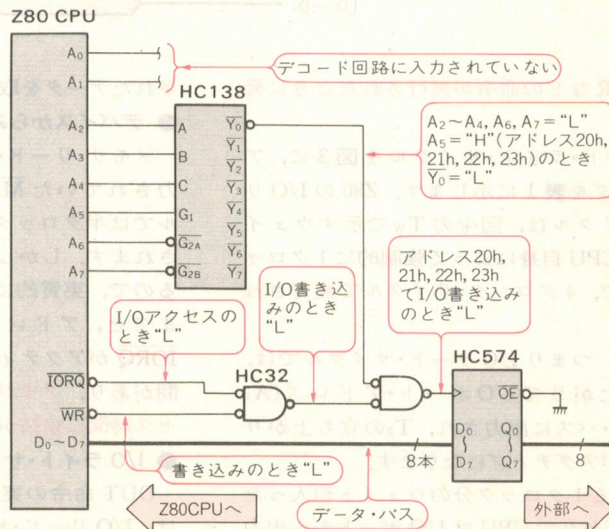
またI/Oデバイスは，$\overline{WR}$クロックの立ち上がりでCPUからデータ・バスに出力されたデータを取り込みます．

アドレス・デコードとは

すでに説明したように，I/Oデバイスの基本はラッチとバッファです．ここではラッチとバッファで構成する非常に簡単なI/Oについて説明します．

● I/Oアドレス・デコード回路

Z80の場合，I/O空間はアドレス・バスA_0〜A_7で指

〈図4〉
8ビット・パラレル出力ポートの回路の例

定できる0～255までの256個です．Z80では本来未定義な16ビットI/Oアドレスを使うならA_0～A_{15}を使いますが，ここでは基本である8ビットI/Oアドレスで説明します．

256個のアドレスの中から，選択したいアドレスに該当するI/Oデバイスだけをアクティブにするのが I/Oアドレス・デコーダです．

I/Oアドレス・デコードにHC138を使い，出力ラッチにHC574を使ったI/Oポートの回路を図4に示します．

すでに説明したように，Z80のI/O空間の選択は$\overline{IORQ}$をLレベルにすることで制御しています．このためI/Oデバイスの選択には$\overline{IORQ}$信号を負論理ANDする必要があります．

この回路では$\overline{IORQ}$と$\overline{WR}$を負論理ANDした$\overline{IOWR}$(I/O書き込み要求)をつくり，これとHC138でデコードしたセレクト信号をANDしてHC574の書き込みクロックにしています．

● **HC574のアドレス**

HC138にはA～C入力と，G_1，$\overline{G_{2A}}$，$\overline{G_{2B}}$の三つのイネーブル端子があります．つまり図4では，A_7とA_6がLレベル，A_5がHレベルのときに，A_2～A_4の状態で$\overline{Y_0}$～$\overline{Y_7}$が選択されます．またA_0とA_1はデコード回路に入力していません．

以上よりHC574は，20h，21h，22h，23hのどのアドレスでも選択することができます．またHC138のイネーブル端子に$\overline{IORQ}$を接続しているわけではないので，HC138の$\overline{Y_0}$端子はI/O命令以外のメモリ・アクセスやリフレッシュのときにもアクティブになることがある点に注意します．

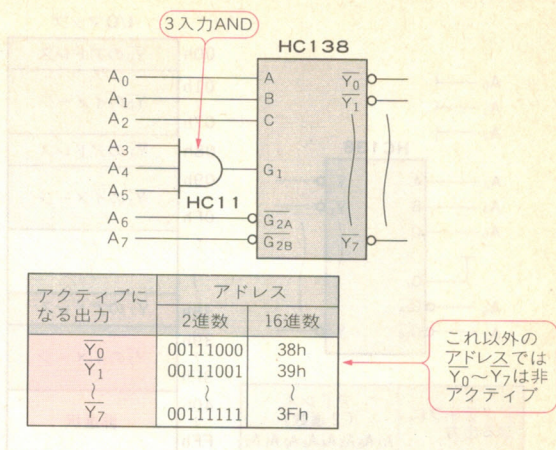
これではI/OデバイスであるはずのHC574が，I/O命令以外のときでも選択されてしまいそうです．しかし$\overline{IOWR}$は$\overline{IORQ}$と$\overline{WR}$がアクティブになるとき，つまりOUT命令を実行したときにアクティブになります．この信号とANDを取っているので，20h，21h，22h，23hのI/Oアドレスに対してOUT命令を実行するときだけ書き込みクロックが出力されるのです．

● **アドレスのフル・デコードとは**

一つのI/Oデバイスに対して複数のアドレスからアクセスできるのは，たしかに気持ち悪いかもしれません．複数のアドレスで同じI/Oが選択されてしまうのは，A_0とA_1をデコード回路に入力していないことに原因があります．

つまりA_0とA_1が“0”でも“1”でもHC138の$\overline{Y_0}$がLレベルになるからです．アドレスをビットで表現すると001000XXbとなります．このXXのことをDon't Careといいます．またこのようなアドレスをイメージ・アドレスともよびます．

〈図5〉8ビットI/Oアドレスをフル・デコードした例

アクティブになる出力	アドレス	
	2進数	16進数
$\overline{Y_0}$	00111000	38h
$\overline{Y_1}$	00111001	39h
～	～	～
$\overline{Y_7}$	00111111	3Fh

どうしてもただ一つのアドレスで選択できなければだめだという場合は，図5のようにA_0とA_1もデコード回路に接続します．これをフル・デコードといいます．

しかしつないでいるI/Oデバイスが数個で，あとから別のI/Oを拡張することもないのであれば，わざわざゲートを使ってフル・デコードする必要はありません．また部品の節約にもなります．

フル・デコードは例えば100個以上ものI/Oデバイスを接続するときに必要になります．Z80のI/O空間は基本的に256バイトですから，一つのI/Oデバイスに四つのアドレスが振られていたのでは，アドレスが足りなくなってしまいます．

フル・デコードされていないI/Oの場合は，一般的にはDon't Careのビットに0を入れたアドレスでアクセスします．

●**アドレス・デコードとI/Oマップ**

デコーダに入力するアドレス線の違いは，I/Oデバイスのアドレスやイメージ・アドレスの発生の違いに現れます．

図6にデコーダ回路に接続するアドレス線の違いによるI/Oアドレスとそのイメージ・アドレスの違いを示します．$\overline{Y_0}$をI/Oアドレス00hでアクティブにしたいなら，図6(a)のように，G_1をHレベルにしてA_3～A_7でデコードします．この場合，01h～07hまで00hのイメージ・アドレスが発生します．$\overline{Y_1}$がアクティブになるのは08hで，09h～0Fhまでがそのイメージになります．そしてアドレス40h～0FFhまでは$\overline{Y_0}$～$\overline{Y_7}$のどれもアクティブにはなりません．

では，$\overline{G_{2A}}$と$\overline{G_{2B}}$に接続しているA_7とA_6を外して入力をグラウンドに落とすとどうなるでしょう［図6(b)］．A_7とA_6を外したのでこのビットもDon't Careになります．しかし$\overline{Y_0}$～$\overline{Y_7}$のアドレスは変化していません．

〈図6〉アドレス・デコードとI/Oマップの例

アクティブになる出力	アドレス(2進数) A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
$\overline{Y_0}$	0 0 0 0 0 × × ×
$\overline{Y_1}$	0 0 0 0 1 × × ×
〜	〜
$\overline{Y_7}$	0 0 1 1 1 × × ×

(a)

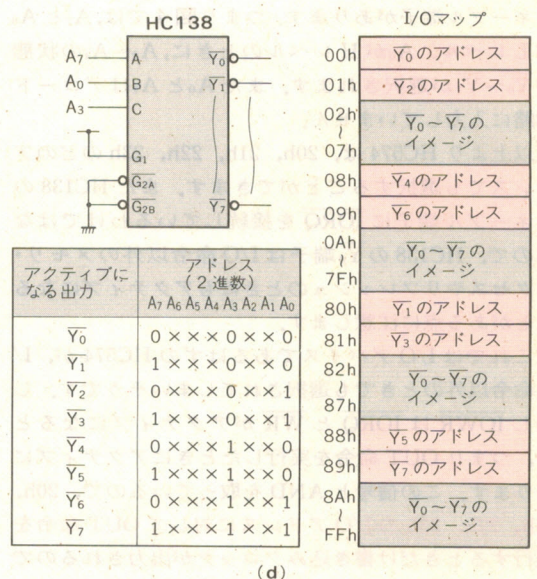
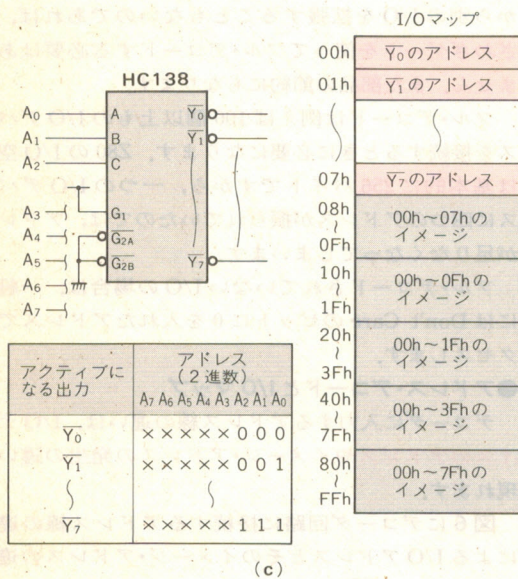
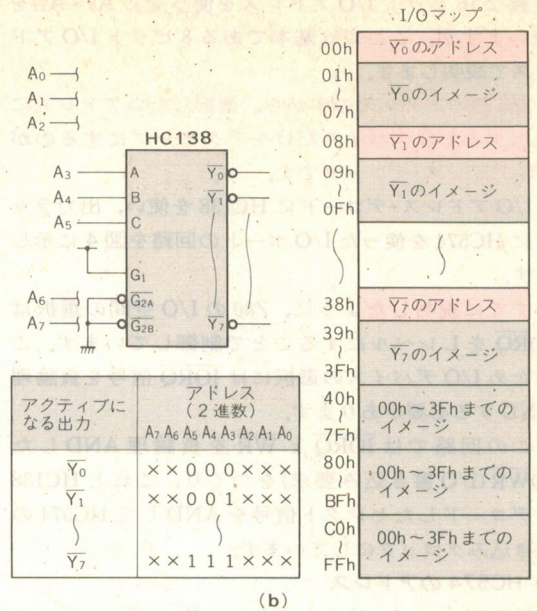
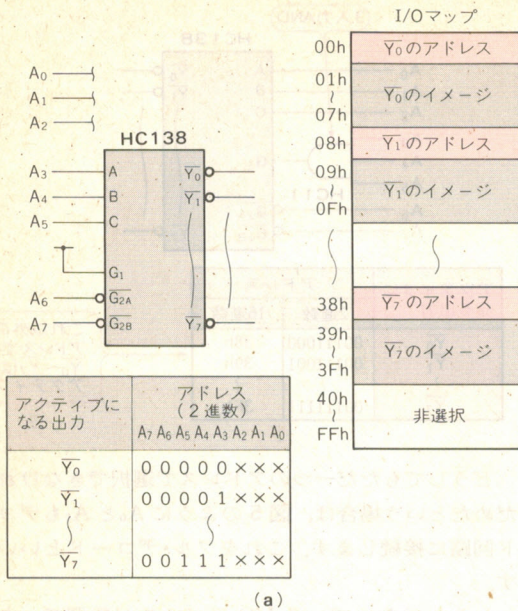
アクティブになる出力	アドレス(2進数) A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
$\overline{Y_0}$	× × 0 0 0 × × ×
$\overline{Y_1}$	× × 0 0 1 × × ×
〜	〜
$\overline{Y_7}$	× × 1 1 1 × × ×

(b)

アクティブになる出力	アドレス(2進数) A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
$\overline{Y_0}$	× × × × × 0 0 0
$\overline{Y_1}$	× × × × × 0 0 1
〜	〜
$\overline{Y_7}$	× × × × × 1 1 1

(c)

アクティブになる出力	アドレス(2進数) A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
$\overline{Y_0}$	0 × × × 0 × × 0
$\overline{Y_1}$	1 × × × 0 × × 0
$\overline{Y_2}$	0 × × × 0 × × 1
$\overline{Y_3}$	1 × × × 0 × × 1
$\overline{Y_4}$	0 × × × 1 × × 0
$\overline{Y_5}$	1 × × × 1 × × 0
$\overline{Y_6}$	0 × × × 1 × × 1
$\overline{Y_7}$	1 × × × 1 × × 1

(d)

しかし図(a)では40h以降のアドレスではすべて非選択になったものが，ふたたび$\overline{Y_0}$がアクティブになります．

このように下位ビットがDon't Careの場合は，同じI/Oが続けて選択されるようにイメージが発生しますが，上位ビットがDon't Careの場合は，それ以下のアドレスを繰り返すようにイメージが発生します．

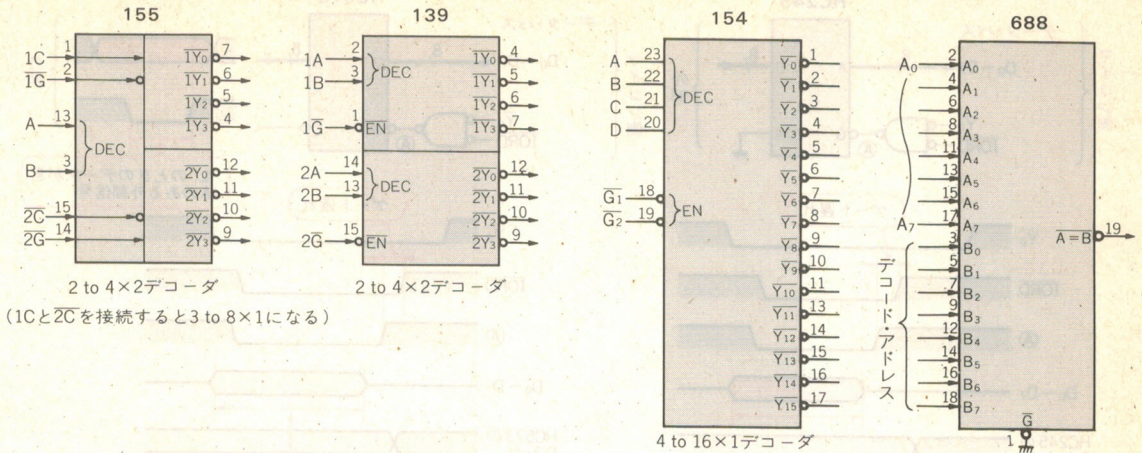
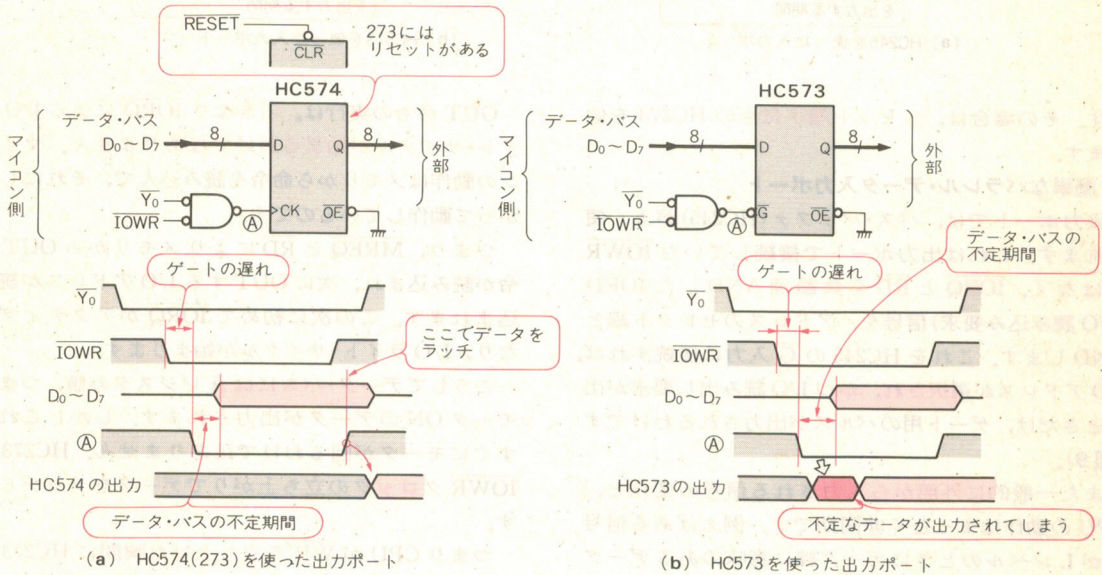
また，いままでの説明ではHC138のA〜C入力端子にはアドレス・バスをA_2〜A_4のように順番に接続していました．もしこの順番が逆，またはA_7をA入力に，A_0をB入力に入れるなどしたらどうなるでしょうか．図6(d)にその例を示しますが，$\overline{Y_0}$〜$\overline{Y_7}$のセレクト線がアドレスに対して順番に選択されず，ばらばらになってしまいます．

もちろんこうしてもアドレスの指定を間違わなければ動作しますが，特に理由のない限り，順番に接続したほうがアドレスのセレクトも順番になりわかりやすくなります．

● HC138以外のデコードIC

アドレス・デコードに使えるICは，HC138以外に

〈図7〉よく使われるデコードIC

(a) HC574(273)を使った出力ポート　(b) HC573を使った出力ポート

〈図8〉パラレル・データ出力ポートの例

も図7に示すようなものがあります．

HC138の出力は8本なので，8個のI/Oデバイスを接続することができます．8個以上のデバイスを接続したいときは，HC154を使ったりデコードICを組み合わせたりします．

またHC688(8ビット・コンパレータ)を使うと，B入力側の状態を変えるだけで任意のアドレスを選択することもできます．

基本的なデータ入出力ポートの設計

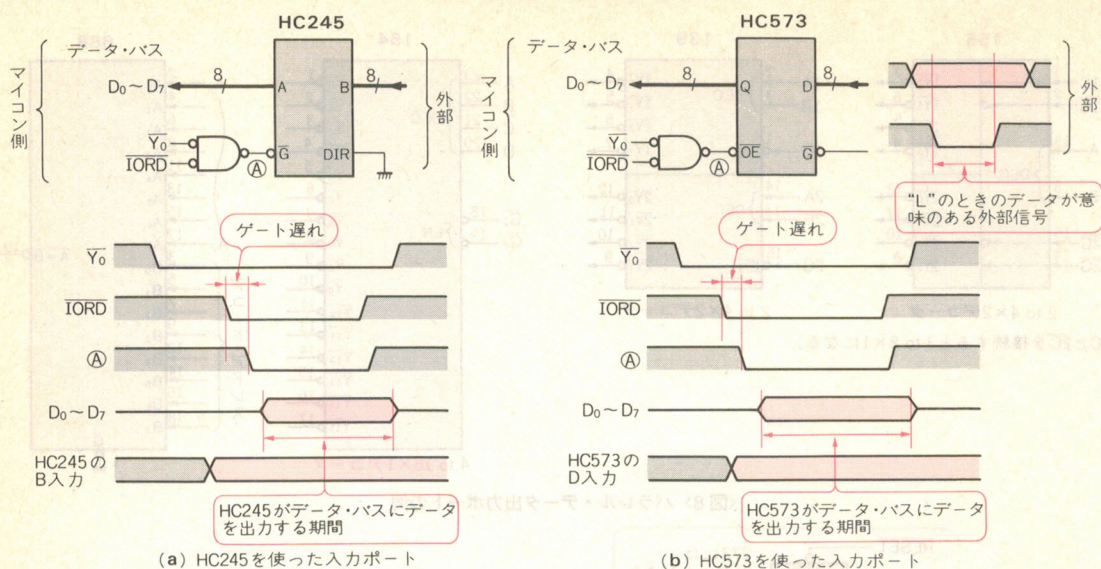
● 簡単なパラレル・データ出力ポート

いちばん基本的な8ビットのパラレル出力ポートとしては，すでに図4に回路例を示しました．このような出力ポートには8ビットのDラッチ(HC574, HC374)がよく使われます．

ここで注意したいのは，トランスペアレント型ラッチであるHC573やHC373は不向きである点です．表1のZ80のI/Oリード/ライト・サイクルをみるとわかるように，$\overline{WR}$の立ち下がりではデータ・バスのデータがまだ確定していません．

HC573などでは$\overline{WR}$が立ち下がった時点でデータ・バスのデータを取り込んで外部に出力してしまうので，図8(b)に示すように不定データが出力されてしまうことになります．

またHC574はリセット端子がなく，電源投入直後は不定データが出力されます．ものによってはリセット直後出力がクリアされていないとまずい場合があり

〈図9〉パラレル・データ入力ポートの例

(a) HC245を使った入力ポート

(b) HC573を使った入力ポート

ます．その場合は，リセット端子付きのHC273を使います．

● 簡単なパラレル・データ入力ポート

入力ポートでは，バス・バッファ(HC245)がよく使われます．これは出力ポートで接続していた$\overline{\mathrm{IOWR}}$ではなく，$\overline{\mathrm{IORQ}}$と$\overline{\mathrm{RD}}$を負論理ANDした$\overline{\mathrm{IORD}}$(I/O読み込み要求)信号を，アドレスのセレクト線とANDします．これをHC245の$\overline{\mathrm{G}}$入力に接続すれば，I/Oアドレスが選択され，かつI/O読み出し要求が出たときだけ，ゲート用のパルスが出力されるわけです(図9)．

また一般的に外部から入力される信号の変化と，CPUの動作はまったく非同期です．例えばある信号線がLレベルのときにデータ線に意味のあるデータが出力される装置からデータを読み込みたいときは，HC573を使いそのデータをラッチします．こうすることで，その装置が常時データを出力しなくても，有効なデータを保持しておくことができ，CPUはいつでもそのデータを読み込むことができます［図9(b)］．

● OUT命令実行時のバスの動作

OUT命令を実行したときの具体的なバスの動きを図10に示します．ここではHC273のQ_1ピン(ビット0)にモータを接続してあるとします．モータはビット0が"1"になると回転します．また電源投入後すぐにモータが回転するとたいへんなので，HC273のクリア端子にリセット信号を接続しています．

プログラムは，ビット0でモータの制御を行うので，Aレジスタに01h(モータのONデータ)を格納し，モータ・コントロールのI/Oアドレス(回路ではアドレス00h)にOUT命令を実行します．

OUT命令の実行は，いきなり$\overline{\mathrm{IORQ}}$によるI/Oライト・サイクルが始まるわけではありません．マイコンの動作はメモリから命令を読み込んで，それにしたがって動作しているのです．

つまり，$\overline{\mathrm{MREQ}}$と$\overline{\mathrm{RD}}$によりメモリからOUT命令が読み込まれ，次にOUTするI/Oアドレスが読み込まれます．この次に初めて$\overline{\mathrm{IORQ}}$がアクティブになり，I/Oライト・サイクルが始まります．

こうしてデータ・バスにはAレジスタの値，つまりモータONのデータが出力されます．しかしこれですぐにモータが回るわけではありません．HC273は$\overline{\mathrm{IOWR}}$クロックの立ち上がりでデータをラッチします．

つまりCPUが$\overline{\mathrm{WR}}$を立ち上げた瞬間にHC273はデータをラッチし，これが初めて外部に出力されモータが回りだすのです．

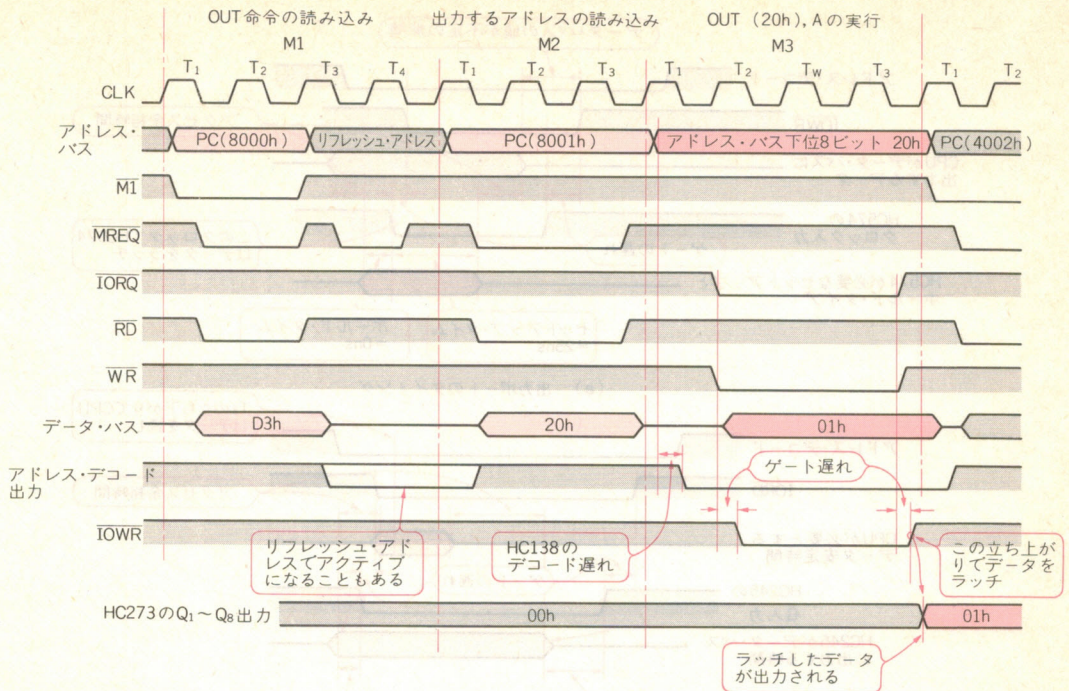
● IN命令実行時のバスの動作

IN命令を実行したときの具体的なバスの動きを図11に示します．ここではHC245のB入力にスイッチを接続してあるとします．

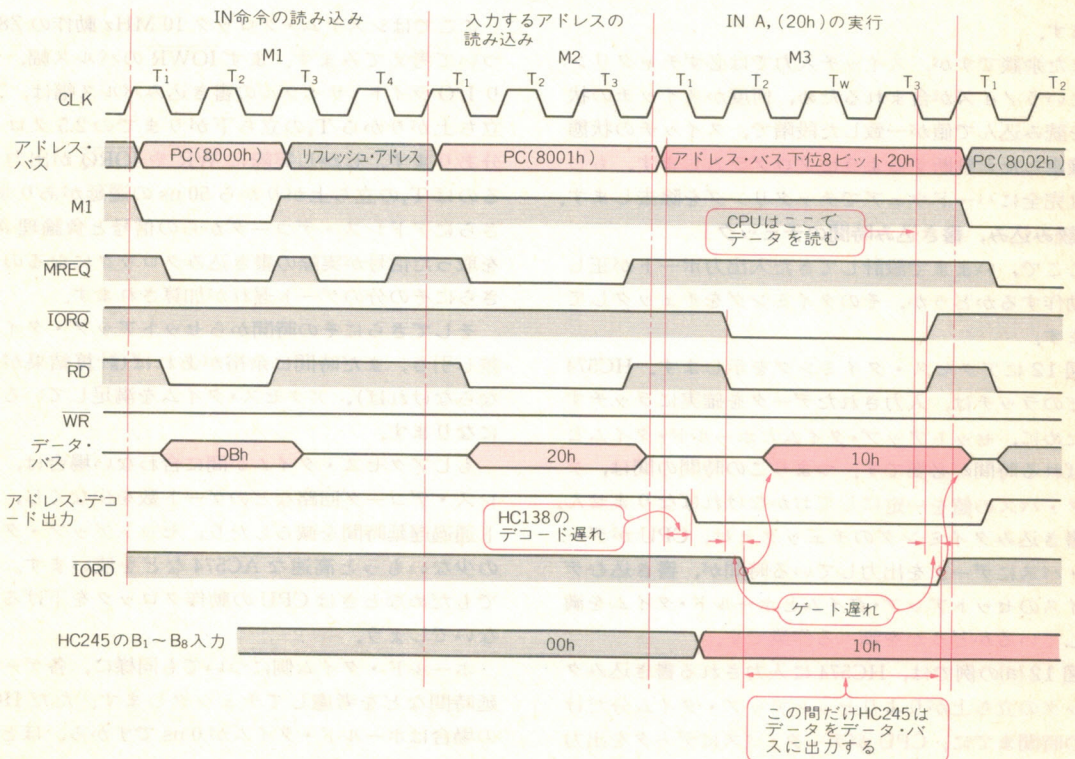
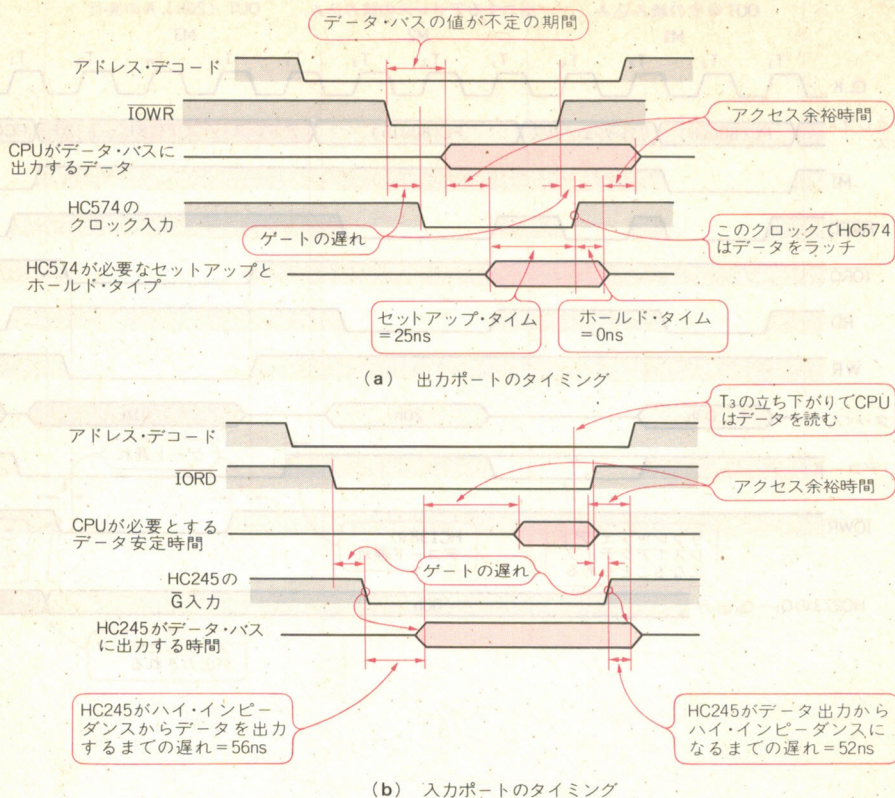
これもOUT命令同様，いきなりI/Oリード・サイクルが始まるのではなく，IN命令とデータを入力するI/Oアドレスをメモリから読み込んだあとに始まります．

I/Oリード・サイクルが始まると，アドレス・バスには指定されたI/Oアドレスが出力され，$\overline{\mathrm{Y_1}}$がアクティブになります．そして$\overline{\mathrm{IORD}}$がLレベルになるので，HC245の$\overline{\mathrm{G}}$入力がアクティブになり，データ・バスにスイッチの状態が出力されます．

CPUはそのデータを読み込み，Aレジスタに格納

〈図10〉OUT命令のバスの動作

〈図11〉IN命令のバスの動作

〈図 12〉 アクセス・タイミングのチェック

(a) 出力ポートのタイミング

(b) 入力ポートのタイミング

します．

また余談ですが，スイッチ入力では必ずチャタリングというノイズが含まれるため，何度かスイッチの状態を読み込んで値が一致した段階で，スイッチの状態が確定したと判断するようにプログラムします．もしくは完全にハードウェアでチャタリングを除去します．

● **読み込み，書き込み時間のチェック**

ここで，いままで設計してきた入出力ポートが正しく動作するかどうか，そのタイミングをチェックしてみます．

図 12 にアクセス・タイミングを示します．HC574 などのラッチは，入力されたデータを確実にラッチするために，セットアップ・タイムとホールド・タイムと呼ばれる時間が必要です．つまりこの時間の間は，データ・バスの値を一定にしておかなければなりません．

書き込みタイミングのチェックとは，CPU がデータ・バスにデータを出力している時間が，書き込むデバイスのセットアップ・タイムとホールド・タイムを満足しているかどうかを調べる作業です．

図 12(a)の例では，HC574 に入力される書き込みクロックの立ち上がりよりセットアップ・タイム分だけ前の時間までに，CPU がデータ・バスにデータを出力しているかどうかがポイントになります．

ここではシステム・クロック 10 MHz 動作の Z80 について考えてみます．まず IOWR のパルス幅，つまり I/O ライト・サイクルの書き込みパルス幅は，T_2の立ち上がりから T_3の立ち下がりまでの 2.5 クロック分あります．しかし実際に WR や IORQ が出力されるのは T_2の立ち上がりから 50 ns の遅延があります．さらにアドレス・デコーダからの信号と負論理 AND を取った信号が実際の書き込みクロックになるので，さらにその分のゲート遅れが加算されます．

そしてさらにその時間からセットアップ・タイムを差し引き，まだ時間に余裕があれば(計算結果が負にならなければ)，アクセス・タイムを満足している結果になります．

もしアクセス・タイムが間に合わない場合は，アドレス・デコーダ回路などのゲート数を少なくし，ゲート通過遅延時間を減らしたり，セットアップ・タイムの少ないもっと高速な AC574 などを使います．それでもだめなときは CPU の動作クロックを下げるしかないでしょう．

ホールド・タイム側についても同様に，各ゲート遅延時間などを考慮してチェックします．ただ HC547 の場合はホールド・タイムが 0 ns ですから，ほとんどの場合問題ないでしょう．

次に図12(b)の読み込みアクセス・タイムについて考えます．これもさきほどのラッチと同様で，CPUがデータを読み込むにはセットアップ・タイムとホールド・タイムの時間だけ，データ・バスの値が一定でなければなりません．

10 MHzのZ80のI/Oリード・サイクルでは，$\overline{RD}$がT_2の立ち上がりから55 ns遅れます．またHC245の読み出しクロックは，CPUの$\overline{IORQ}$と$\overline{RD}$，そしてアドレス・デコーダの出力を負論理ANDした信号になるので，これらのゲート遅延時間も加算されます．

そしてHC245に読み出しクロックが入力されてから，実際にデータ・バスにデータが出力されるまでの遅延時間も考慮に入れる必要があります．以上を計算しても，CPUが必要とするセットアップ・タイム以上の時間があればだいじょうぶです．

入力専用と出力専用のポート

● 出力専用ポートで読み込みをしたら

出力にHC574を入力にHC245を使ったいままでの例では，それぞれ別々のアドレスに対して出力ポートと入力ポートを割り振っていました．

もしここで，HC273のI/Oアドレスに対して入力動作をしたらどうなるでしょう．$\overline{Y_0}$はアクティブになりますが，$\overline{IOWR}$がアクティブにならないので，HC273に書き込み動作は行われません．また当然データを出力するデバイスも存在しないわけですから，CPUは誰もデータを出力していないデータ・バスから無意味なデータを読み込みます．

逆にHC245のI/Oアドレスに対して出力動作をしても，$\overline{Y_1}$はアクティブになりますが，$\overline{IORD}$がアクティブにならないので，HC245はデータ・バスにデータを出力しません．またCPUが出力したデータは誰も受け取ることなく消えてしまいます．

● 出力専用ポートではデータを保存する

このように，HC273だけを接続したI/Oアドレスは出力専用で，そのI/Oアドレスに出力したデータを再び読み込むことはできません．このような出力専用ポートにデータを出力するときは，出力したデータを後で調べられるように，RAMにワーク・エリアを設定して出力したデータを保存しておくという方法を取ります．

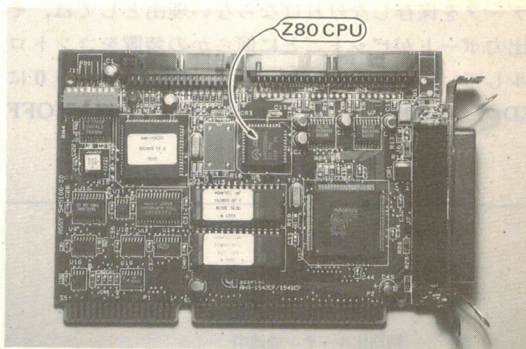
Z80はまだまだ現役

DOS/Vユーザでもある筆者は，ようやく昨年の年末にSCSIボードで標準となったアダプティック社のAHA1542CFとCD-ROMドライブを購入しました．ふとボードを見ると，なんとZ80CPUが搭載されているのです(**写真A**)．

〈写真A〉Z80が使われているSCSIボード

Z80もCMOS化され高速になり，日本ではASSPであるZ84C011やZ84C015がよく売れているそうです．Z80もなんだかんだ言われながら，たいていのシステムは実現できる能力をもっています．メモリの問題も少々めんどうですが，バンク切り替えで何とかなります．高速処理が必要な部分にはDSPを速いCPUの代わりに使おうとしているくらいで，ほとんどのことは筆者もいまだにZ80です．

Z80を使う魅力の一つに開発ソフトの安さがあります．30半ばになった筆者も，最近でははんだごてを持つよりソフトを作るほうが楽になりました．ソフトもFORTRAN，BASIC，アセンブラ(Z80，8085)，Cと移ってきました．しかしC++にはなかなか移行できませんが…．

PC8001の拡張ボックスにPPIだけでボードを作り，漢字ROMを接続してBASICのINとOUT命令だけで動作させたときの感動が，マイコンとのつき合いの始まりでした．BASICの動作速度の遅さがアセンブラを身につけさせ，PALライタがマイコン・ボードを身につけさせてくれました．

言語がCになっただけで，DOS/VマシンでもPC8001と同様のことができると知り，ドータ・ボードを買ってやってみるところです．EDLINとPPIの達人を自称する筆者も，まだまだZ80と同様で現役のようです．

〈村田浩義〉

〈図 13〉 同じ I/O アドレスを使った入力，出力ポート例

使用例 ①

デバイスＡに書き込むコマンド

デバイスＡから読み込むステータス

入力と出力の両方の動作があるI/Oのときは，同じアドレスに対して読み込みと書き込みができるとわかりやすい．

使用例 ②

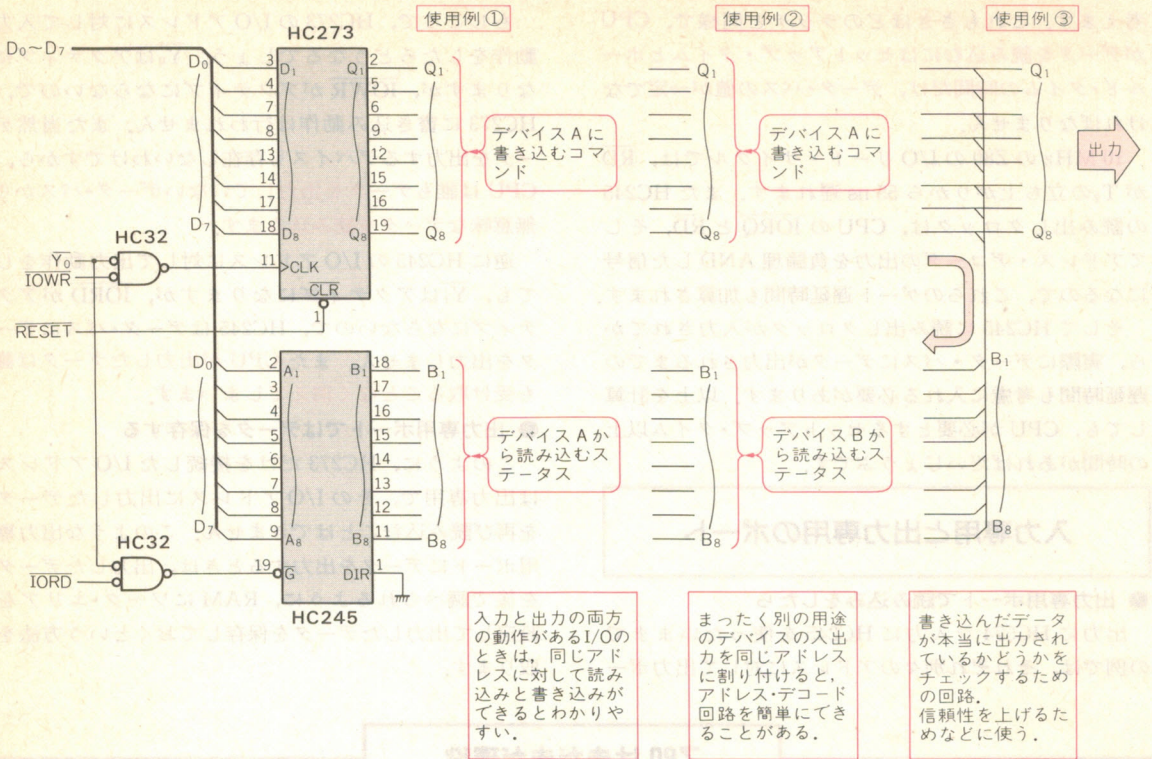
デバイスＡに書き込むコマンド

デバイスＢから読み込むステータス

まったく別の用途のデバイスの入出力を同じアドレスに割り付けると，アドレス・デコード回路を簡単にできることがある．

使用例 ③

出力

書き込んだデータが本当に出力されているかどうかをチェックするための回路．信頼性を上げるためなどに使う．

データを保存しなければならない理由としては，その出力ポートがビットごとに何らかの装置をコントロールしているときなどのためです．例えばビット 0 に LED の ON/OFF を，ビット 7 にモータの ON/OFF という意味をもたせたとします．いま LED が ON，モータが OFF している状態で，モータを ON したいとします．単にビット 7 を 1 にして 80h をポートに出力したらどうなるでしょうか．ビット 0 まで OFF になり，LED が消えてしまいます．

〈リスト 1〉 出力専用ポートでのデータ保存法

```
CTRL_A: EQU     80H     ;コントロール・ポート
                        ;bit0  "1" LED ON
                        ;bit7  "1" MOTER ON

;       初期化ルーチンの途中
        LD      A,0
        OUT     (CTRL_A),A        ;出力クリア
        LD      (CTRL_A_SAVE),A   ;出力データ保存

;       LED ONルーチン
LED_ON: PUSH    AF
        LD      A,(CTRL_A_SAVE)
        OR      1                 ;bit 1 ON
        OUT     (CTRL_A),A        ;LED ON
        LD      (CTRL_A_SAVE),A   ;保存
        POP     AF
        RET

;       LED OFFルーチン
LED_OFF:PUSH    AF
        LD      A,(CTRL_A_SAVE)
        AND     0FEH              ;bit 1 OFF
        OUT     (CTRL_A),A        ;LED OFF
        LD      (CTRL_A_SAVE),A   ;保存
        POP     AF
        RET
```

このようなときのために，出力専用ポートに出力したデータを保存しておき，操作したいビットに対してのみ ON/OFF 操作をしたあとそのデータを出力すれば，ほかのビットの状態に影響を与えないでビットのコントロールを行うことができます(**リスト 1**)．

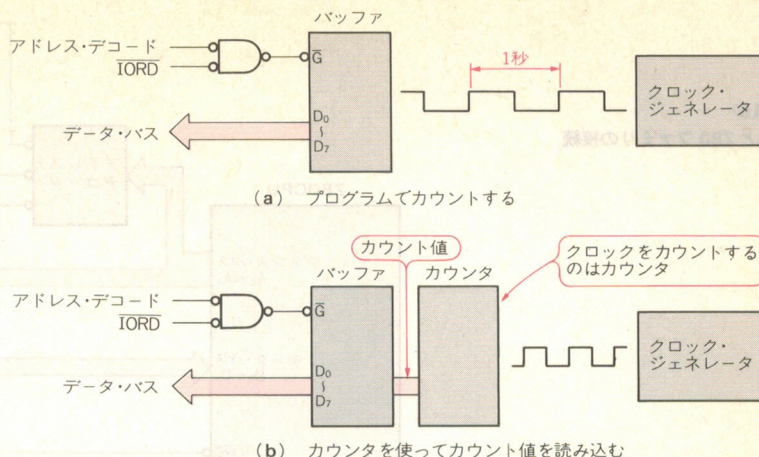
● **同一アドレスに入出力ポートを割り当てる**

$\overline{Y_0}$ と $\overline{IOWR}$ を AND した信号を HC273 の CK 端子へ，そして $\overline{Y_0}$ と $\overline{IORD}$ を AND した信号を HC245 の $\overline{G}$ へ接続するとどうなるでしょう(図 13)．つまり入力と出力を同一 I/O アドレスに割り振るのです．

この I/O アドレスに対して出力動作をすると，データ・バスのデータが HC273 に書き込まれます．また入力動作をすると，データ・バスに HC245 のデータが出力されます．

ある一つの装置のコントロールのために，書き込みと読み出しの必要があるときは，このように同じアドレスに読み込みと書き込みを割り付けるとわかりやすくなります．

また場合によっては I/O アドレス・デコード回路の

〈図14〉
時計機能を実現する

ゲート数節約のために，まったく別の意味をもつI/Oデバイスのアドレスを共有して，出力ポートと入力ポートを同一アドレスにすることもあります．

同じアドレスでありながら，出力は装置Aのコントロールの機能を，入力では装置Bのステータスを読み込むという使い方も可能です．

またHC273のQ_1〜Q_8の出力をHC245のB_1〜B_8に接続すると，出力でHC273に書き込んだデータをHC245から読み込むことが可能になります．これでワーク・エリアにいちいちデータを保存しなくても，出力したデータを読み込むことができます．

I/Oコントローラとは何か

● ラッチやバッファ以外のI/Oデバイス

いままでは非常に簡単なラッチやバッファによるI/O回路について説明しました．しかし実際にマイコンをコントローラとして使う場合は，接続する外部装置の形態や扱うデータの形式によって，それぞれ特化したI/Oデバイスがあると便利です．

例えばマイコンに時計機能を実現させるために，いままで説明してきたバッファでこれを作るとすると，1秒周期のクロックをバッファから読み込んで，プログラムでカウントしなければなりません［図14(a)］．

もう少し頭をひねり汎用ロジックICのカウンタを使い，そのカウント・データをバッファを介して読み込めば，プログラムでクロックをカウントする必要はありません［図14(b)］．

図をみると最低バッファとカウンタが必要で，カウンタの値をクリアしたいなどの機能を入れるとさらに回路は複雑になります．これらクロックのカウント機能を一つのチップに詰め込んだLSIがあると非常に便利です．このクロックのカウントに特化したLSIがCTCと呼ばれるLSIなのです．

〈表2〉 **Z80ファミリの種類**

PIO(パラレル出力コントローラ)
パラレルで外部装置とやりとりをする基本的なI/O． A-Dコンバータの制御やEPROMライタの制御によく使われる．
SIO(シリアル入出力コントローラ)
シリアルで外部装置とやりとりをするI/O． RS-232-Cのコントローラとして使われる．
CTC(カウンタ・タイマ・コントローラ)
時間やクロックを数えるカウンタ． 数を数えるものに使われる．
DMA(ダイレクト・メモリ・アクセス・コントローラ)
メモリやI/O間でCPUを介さずに直接データの転送を行うためのコントローラ． 最近のZ80システムでは使われなくなってきた．

● マイコンで汎用的に使われるコントローラ

このカウンタ・タイマ用のコントローラであるCTC(カウンタ・タイマ・コントローラ)以外にも，ラッチやバッファの機能を一つで実現でき，プログラムによって出力や入力を設定できるPIO(パラレルI/O)，またRS-232-Cなどのシリアル通信用にはSIO(シリアルI/O)があります．

これらCTC，PIO，SIOはマイコン・システムにおいては汎用的に使われる機能で，非常にたくさんの種類のLSIがあり，それぞれに細かい機能の違いや特徴があります．

よって機能や使用するCPUとの接続のしやすさなどから選択します．ここではCPUとしてZ80を取り上げているので，Z80に接続しやすいコントローラということになります．

Z80ファミリLSIの接続

● Z80ファミリLSIとは

Z80の特徴の一つに，Z80用周辺コントローラLSI

〈図 15〉
Z80 と Z80 ファミリの接続

の充実があります．これには Z80PIO（パラレル I/O），Z80CTC（カウンタ・タイマ），Z80SIO（シリアル I/O）があります（表 2）．これらはさすが Z80 ファミリを名乗っているだけあって，Z80CPU と非常に簡単に接続することができます．

また後述するモード 2 割り込みを使うための仕掛けが，これらファミリ LSI には内蔵されているので，複雑で強力な Z80 のモード 2 割り込みシステムを，周辺 LSI を接続するだけで実現できます．

● Z80 ファミリと Z80CPU の接続

Z80 周辺 LSI は，基本的には Z80CPU 専用に作られている周辺コントロール LSI 群です．当然 CPU とも簡単に接続できるように考えられています．

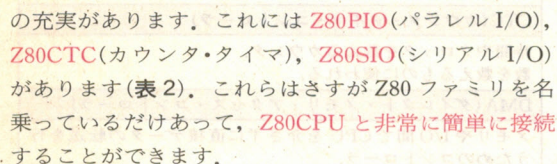



図 15 に Z80 ファミリを接続するときの概念図を示します．Z80 ファミリは CPU と同期を取るために CPU と同じシステム・クロックを必要とします．また $\overline{IORQ}$，$\overline{RD}$，$\overline{M1}$，$\overline{INT}$ の各制御線を接続します．

〈図 16〉 Z80PIO のリセットのかけ方

ラッチやバッファの例では，$\overline{IORQ}$ と $\overline{RD}$ を AND した $\overline{IORD}$ などの信号を使いましたが，Z80 周辺 LSI では，直接 $\overline{IORQ}$ を接続する端子があるので，必ず I/O 空間に対するアクセスで選択されるようになっています．

● $\overline{WR}$ 端子は接続しない

Z80 周辺 LSI は $\overline{WR}$ 端子をもっていません．これでは書き込みができないように思えます．

しかしタイミングをよく考えてみれば，$\overline{RD}$ と $\overline{WR}$ が同時にアクティブになることはあり得ません．そこで，$\overline{IORQ}$ がアクティブになったとき，$\overline{RD}$ が L レベルなら読み込み，H レベルなら書き込みであると判断しているのです．

このように Z80 ファミリは，ピン数の都合から内部で $\overline{WR}$ を作っているので，$\overline{WR}$ 信号を接続しなくても大丈夫なのです．ただし例外的に Z80DMA には $\overline{WR}$ 端子があります．

● $\overline{M1}$ の重要性

$\overline{M1}$ も Z80 周辺 LSI にとって非常に重要な線です．Z80 周辺 LSI は常時データ・バスを監視して，CPU が割り込み処理からメイン・ルーチンに戻るときの RETI 命令を実行するかどうかをチェックしているからです．

また CPU が割り込みベクタを読み込むときに，$\overline{M1}$ と $\overline{IORQ}$ をアクティブにします．周辺 LSI はこ

の信号が両方Lレベルになったとき割り込みベクタを出力します．割り込みに関する詳しい解説は第6章を参照してください．

● Z80PIOのリセット

また PIOだけは40ピンのパッケージに収めるために，$\overline{\mathrm{RESET}}$ 信号の入力端子を省略し，$\overline{\mathrm{M1}}$ ピンを $\overline{\mathrm{RESET}}$ 信号と兼ねています．

Z80PIOは内部にパワーONリセット回路を内蔵していますが，外部より強制リセットするときは，$\overline{\mathrm{IORQ}}$ と $\overline{\mathrm{RD}}$ をHレベルにして $\overline{\mathrm{M1}}$ 信号を2クロック・サイクル以上Lレベルにします．$\overline{\mathrm{M1}}$ はオペコード・フェッチ時にLレベルになる端子で，第1章で説明したようにオペコード・フェッチ・サイクルは4クロックで，このうち $\overline{\mathrm{M1}}$ が出力されるのが2クロック以内です．

つまり2クロック以上Lレベルが続いたときは，オペコード・フェッチ・サイクルではないと判断できるわけです．

よってZ80PIOの $\overline{\mathrm{M1}}$ 入力へは，CPUからの $\overline{\mathrm{M1}}$ とリセット回路からのリセット信号の負論理ORをとった信号を入力します(図16)．

● アドレス・デコードの処理

残る信号線はアドレス・バスの処理です．Z80周辺LSIにはチップ・セレクト制御として $\overline{\mathrm{CE}}$ 端子があります．またLSI内部にコマンド用やステータス用など複数のポートやレジスタがあります．これらの選択のために，B/$\overline{\mathrm{A}}$，またはC/$\overline{\mathrm{D}}$ などのアドレス線があります．

ここでは例としてZ80PIOの接続についてみてみます．

I/Oアドレスのデコードにはいままでの例と同様HC138などを使います．しかしどのアドレス線でデコードするかはちょっと考えなくてはなりません．

Z80PIOは8ビットのパラレル・ポートを2チャネルもっています．このポートの選択のために B/$\overline{\mathrm{A}}$ というチャネル選択ビットがあります．またそのポートを入力にするか出力にするかなどのコントロール用のレジスタ・ポートと，実際にデータを入出力するためのデータ・ポートを選択する C/$\overline{\mathrm{D}}$ があります．

つまりZ80PIOは一つのLSIで四つのアドレスを必要とするコントローラなのです．プログラムで制御する場合はこれら四つのアドレスが，チャネルAのデータ・アドレス，チャネルAのコントロール・アドレス，チャネルBのデータ・アドレス，チャネルBのコントロール・アドレスと順番に並んでいたほうがわかりやすいでしょう．

このようにアドレスをマッピングするためのデコード回路を図17に示します．この四つのアドレスを順番に並べるためには，C/$\overline{\mathrm{D}}$ に A_0を，$\overline{\mathrm{A}}$/B に A_1を接続すればOKです．そして $\overline{\mathrm{CE}}$ に HC138のセレクト信号を接続します．このHC138に入力するアドレス線と，出力 $\overline{Y_0}$～$\overline{Y_7}$のどれを $\overline{\mathrm{CE}}$ につなぐかによって，

I/Oアクセス時にウェイトを入れる回路

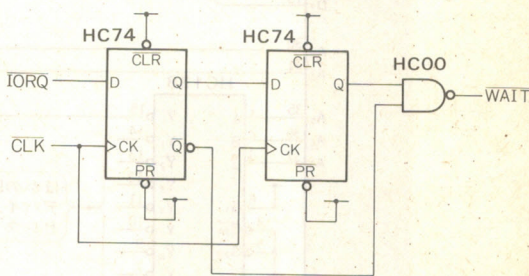
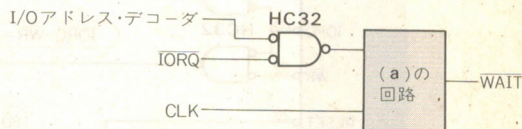
マイコンではI/Oデバイスのアクセス速度はメモリのそれと比較して遅いのが一般的です．システムの動作クロックを決めるとき，遅いデバイスに合わせるのというやり方もありますが，これでは命令の処理速度まで遅くなってしまいます．

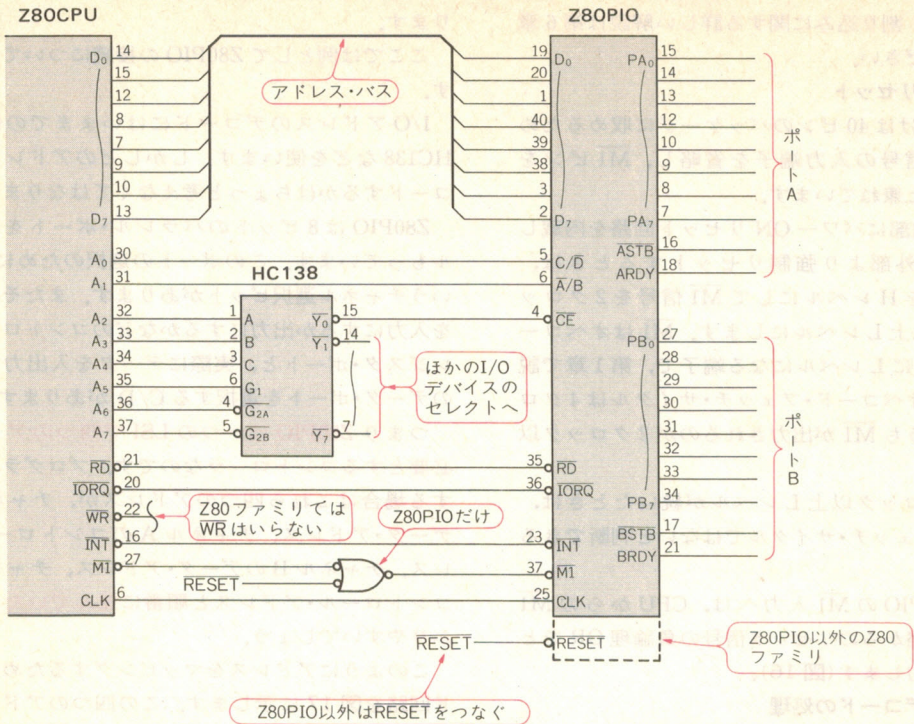
そこで，遅いI/Oアクセスのときだけ CPUに「ちょっと待った！」をかけられる回路があると便利です．これがウェイト回路です．

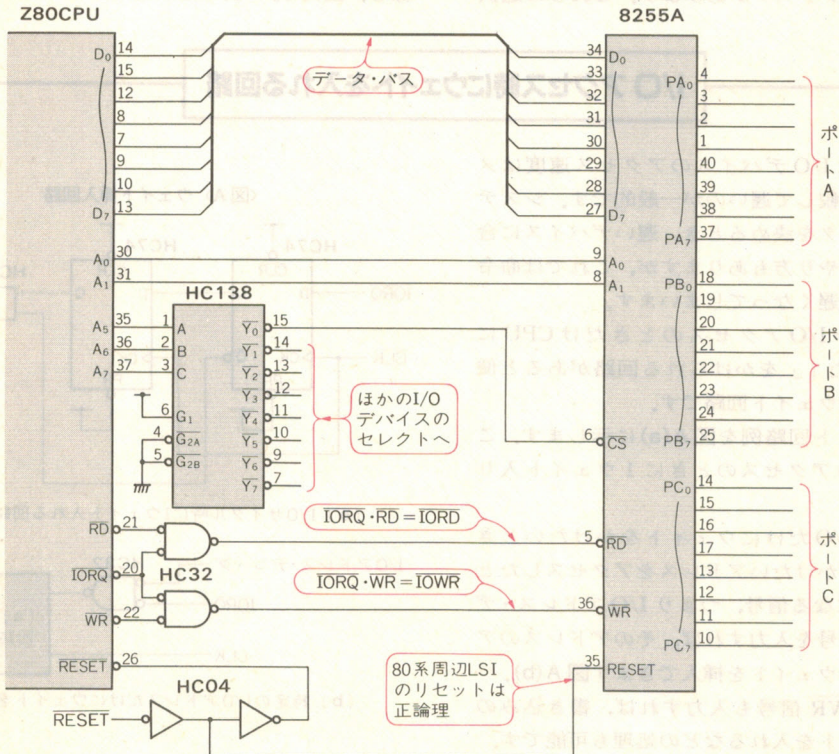
簡単なウェイト回路例を図A(a)に示します．この回路ではI/Oアクセスのときに1ウェイト入ります．

また特定のI/Oだけにウェイトをかけたいときは，ウェイトをかけたいアドレスをアクセスしたときにLレベルになる信号，つまりI/Oアドレス・デコーダの出力信号を入力すれば，そのアドレスのアクセス時にだけウェイトを挿入できます図A(b)．

またさらに $\overline{\mathrm{WR}}$ 信号も入力すれば，書き込みのときだけウェイトを入れるなどの処理も可能です．

〈図A〉ウェイト挿入回路

(a) I/Oサイクル時に1ウェイト入れる回路

(b) 特定のI/Oアドレスだけにウェイトを入れる

〈図 17〉 Z80PIO を例にした Z80 と Z80 ファミリの接続回路

〈図 18〉 8255A を例にした Z80 と 80 系周辺 LSI の接続回路

Z80CPU
8255A
データ・バス
HC138
ほかのI/O
デバイスの
セレクトへ
$\overline{IORQ}\cdot\overline{RD}=\overline{IORD}$
HC32
$\overline{IORQ}\cdot\overline{WR}=\overline{IOWR}$
HC04
$\overline{RESET}$
80系周辺LSI
のリセットは
正論理
ポートA
ポートB
ポートC

Z80PIOの割り振られるアドレスが決まります.

図の例ではA_5〜A_7もHC138に入力しているので，Z80PIOのアドレスは20h，21h，22h，23hの四つとなり，8ビットをフル・デコードしていることになります.

80系周辺LSIの接続

● 8255Aや8251Aをつなぐ

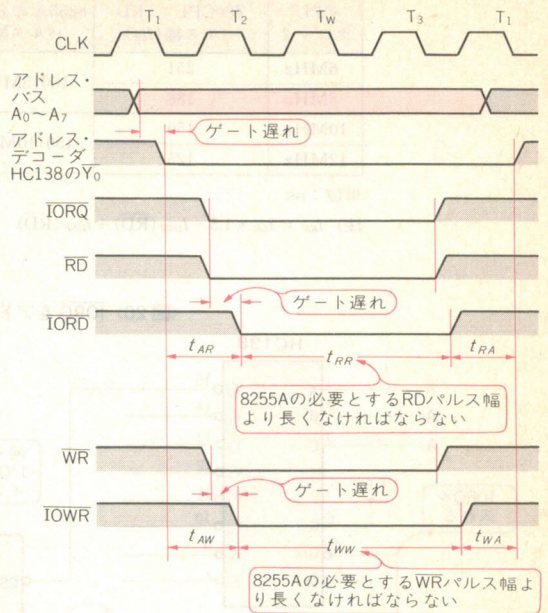
Z80マイコンは8080Aの改良版として登場しましたが，Z80周辺LSIは当時としては斬新で使い方が難しかったため，従来からの80系周辺LSIが依然として使われました.

いまだにPPI(8255A)，USART(8251A)は，Z80マイコン・システムでもお馴染みの周辺LSIです．ちなみに，PPIやUSARTはパソコンでもよく使われています.

80系周辺LSIに対するアクセスは，メモリと同じように$\overline{RD}$，$\overline{WR}$，$\overline{CS}$信号で行います．また内部をクリアするRESET信号はZ80ファミリと極性が反対で，Hレベルでアクティブです.

● Z80と80系周辺LSIの接続

80系周辺LSIとして8255Aをとりあげ，図18にZ80との接続例を示します．8255AにもZ80PIOと同様ポート選択やモード設定のためにアドレスがいくつかあります．そこでアドレス・バスのA_0，A_1を接続します.

〈図19〉Z80と8255Aを接続したときのアクセス・タイミング

80系周辺LSIとZ80周辺LSIとの一番の違いは，$\overline{IORQ}$信号の入力ピンがないことです．このため，$\overline{RD}$，$\overline{WR}$両方に，もしくは$\overline{CS}$のどちらかに必ず$\overline{IORQ}$を接続しなければなりません.

Z80七不思議 I/O編

Q：I/O空間のアドレスを16ビットにしなかったのはなぜ？

A：8080では，I/O空間は256バイトなので，それに合わせてこの大きさになったと思われます．たしかに正式にサポートしているI/O空間は256バイトしかありませんが，Z80CPUではI/O命令を大幅に拡張し，間接I/Oアドレッシングや，I/Oに対するブロック転送命令もサポートするようになりました．直接アドレッシングでは8ビットのI/Oアドレスしか指定できませんが，拡張された命令の中には間接的に上位8ビットの部分を指定することができるものがあり，その命令を使うことにより16ビット，64KバイトのI/O空間を使用することもできます.

もっともZ80の開発の時点では，256バイト以上のI/Oを必要とするアプリケーションがほとんどなかった，というのが真実のような気がしますが.

Q：I/O命令時に自動的に1ウェイト入るのはなぜ？

A：Z80周辺デバイスは，そのI/Oサイクル，RETIサイクル，割り込み応答サイクルを検出する必要があり，そのためCPUからの制御線をクロック・エッジでサンプルし，次のサイクル以降で動作をするようになっています.

Z80CPUでは，ご存じのように$\overline{IORQ}$や$\overline{RD}$，$\overline{WR}$はT_2の立ち上がりで変化しますので，通常のメモリ・サイクルと同じタイミングだとウェイトを入れないと苦しかったから，という理由からではないかと思います．また，Z80CPUが開発された当時の周辺デバイスのスピードがそんなに速くなかったこともあるでしょう.

〈信垣育司〉

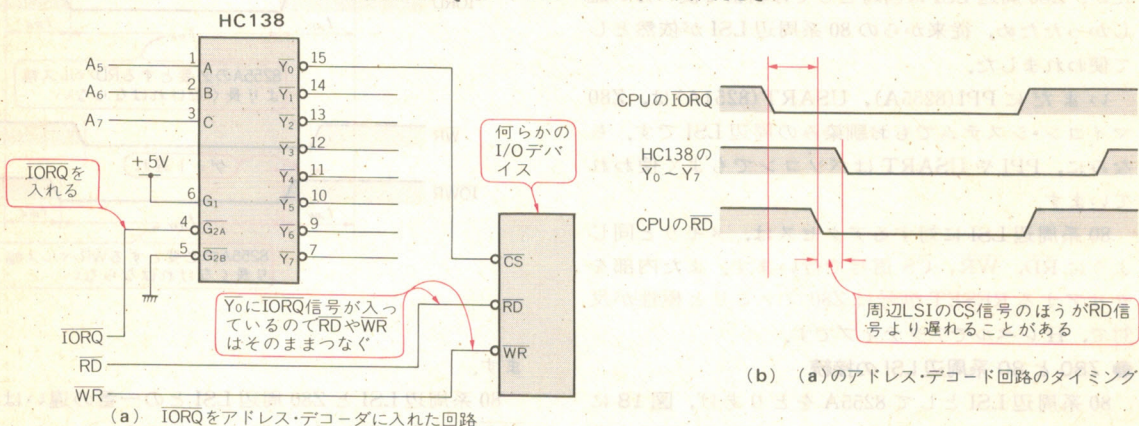
〈表3〉Z80に8255Aを接続したときのアクセス・タイミングの計算

CPU クロック	Z80CPUの $\overline{RD}$ パルス幅(t_{RR})	8255Aの必要とするパルス幅(t_{RR})	Z80CPUの $\overline{WR}$ パルス幅(t_{WW})	8255Aの必要とするパルス幅(t_{WW})
6MHz	251	160(5MHz版)	256	120 (5M/10MHz版)
8MHz	188		193	
10MHz	150	150(10MHz版)	155	
12MHz	125		120	

単位：ns

注）$t_{RR}' = t_{CK} \times 1.5 - t_{DCR}(\overline{RD}) + t_{DCF}(\overline{RD})$　　$t_{WW}' = t_{CK} \times 1.5 - t_{DCR}(\overline{WR}) + t_{DCF}(\overline{WR})$

〈図20〉$\overline{IORQ}$をアドレス・デコーダに入れたとき

（a）$\overline{IORQ}$をアドレス・デコーダに入れた回路　（b）（a）のアドレス・デコード回路のタイミング

回路例では$\overline{IORQ}$信号を$\overline{RD}$信号と$\overline{WR}$信号で負論理ANDした$\overline{IORD}$と$\overline{IOWR}$を，8255Aの$\overline{RD}$と$\overline{WR}$に接続しています．通称インテル結線といいます．また$\overline{CS}$にはHC138によるデコード出力を接続します．

● アクセス・タイミングの計算例

Z80周辺LSIとZ80CPUの接続は，同じZ80ファミリですから動作周波数の同じバージョンを使えば基本的には何も問題ありません．Z80CPUが8MHz版であるなら，Z80PIOも8MHz版を使います．動作クロックが異なるバージョン同士ではアクセスが追いつかず，問題があります．

アドレス・デコードも，HC138などを普通に使う限りではまず問題ありません．よほど複雑で変なデコードをしないかぎり，ゲートの遅れはあまり気にしなくてもだいじょうぶです．

ここではZ80CPUと8255Aなどの80系周辺LSIとを接続した場合です．

I/Oポートからのデータ入力時は$\overline{CS}$と$\overline{RD}$信号が，データ出力時は$\overline{CS}$と$\overline{WR}$信号がアクティブになります．ただしそのタイミングに注意します．

一番問題になるのは，8255Aの必要とする$\overline{RD}$信号と，$\overline{WR}$信号のパルス幅です．Z80と8255Aをつなげたときの$\overline{RD}$信号と$\overline{WR}$信号のパルス幅との関係をまとめると，図19と表3のようになります．表をみると，12MHzのZ80では10MHz動作の8255Aの$\overline{RD}$パルス幅を満足できないことがわかります．

ここでアドレス・デコードを図20のような回路にして，$\overline{CS}$に$\overline{IORQ}$を入れた場合を考えてみます．HC138のゲート遅れによって，$\overline{RD}$や$\overline{WR}$より$\overline{CS}$が遅れることが考えられます．このようなデコーダによる回路がたまに見受けられますが，正確にはタイミングに問題があります．

$\overline{CS}$でイネーブルをかけてから$\overline{RD}$や$\overline{WR}$で読み書きするタイプのI/Oデバイスでは，この点に注意する必要があります．

ただし，デバイスの中にはμPD71055(NEC)のように，$\overline{CS}$より$\overline{RD}$が先に立ち下がっても動作を保証しているものもあります．

◆参考文献◆

(1) 鈴木八十二，村田浩義；CMOSマイコンZ80を用いたシステム設計，日刊工業新聞社．

（トランジスタ技術1994年6月号に加筆，修正）

電源ONからマイコンを動作させるまでの回路

クロック&リセット周辺回路の設計

野口智樹

Z80 の電源とクロック回路

この章では Z80 を動作させるために必要な電源やクロック，そしてマイコンの動作のすべてのスタートであるリセット回路について説明します．

● Z80 の電源

表 1 に Z80 の電源特性を示します．CMOS 版では HALT 命令の実行とクロックを停止させることで，スタンバイ・モードと呼ばれる低消費電力モードを備えています．このときの消費電流は 10 μA と非常に小さく，電池動作の場合などに非常に有利になります．

〈表 1〉 CMOS 版 Z80 の電源特性

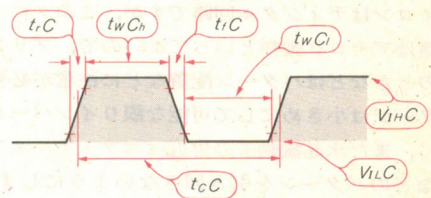
記号	名称		最小	最大	条件	単位
V_{CC}	電源電圧		+4.50	+5.50		V
V_{IL}	入力 “L” 電圧		−0.3	0.8		V
V_{IH}	入力 “H” 電圧		2.2	V_{CC}		V
V_{OL}	出力 “L” 電圧			0.4	I_{OL}=2.0mA	V
V_{OH1}	出力 “H” 電圧		2.4		I_{OH}=−1.6mA	V
V_{OH2}	出力 “H” 電圧		V_{CC}−0.8		I_{OH}=−250μA	V
I_{CC1}	消費電流	4MHz 動作		20		mA
		6MHz 動作		30	V_{CC}=5V	mA
		8MHz 動作		40	V_{IH}=V_{CC}−0.2V	mA
		10MHz 動作		50	V_{IL}=0.2V	mA
		20MHz 動作		100		mA
I_{CC2}	スタンバイ時消費電流			10	CLK=0	μA

● 基本的なクロック発生回路

図 1 に Z80 のクロック入力波形を，**表 2** にクロック入力特性を示します．また**図 2** にいちばん基本的なクロック発生回路を示します．使用した水晶振動子の周波数でクロックが作られます．

表 2 をみると，クロックの H レベルと L レベルは同じ時間となっています．つまりデューティ比が 1：1 のクロックが必要だということです．**図 2** (a)の回路では場合によってはデューティ比 1：1 の波形が得られないことがあるので，図(b)のように間にフリップフロップを入れて 1/2 に分周し，正確なデューティ比 1：1 のクロックを得るようにします．当然この場合は，水晶振動子の発振周波数を 2 倍にしないと，図(a)と同じクロック周波数が Z80 に供給されません．

図 3 に実用的な発振回路を示します．図(a)の回路

〈図 1〉 Z80 のクロック入力波形

〈表 2〉 CMOS 版 Z80 のクロック入力特性

記号	項目	6MHz 版 最小	6MHz 版 最大	8MHz 版 最小	8MHz 版 最大	10MHz 版 最小	10MHz 版 最大	20MHz 版 最小	20MHz 版 最大	単位
T_{CC}	クロック周期	162	DC	125	DC	100	DC	50	DC	ns
t_{wCh}	クロック “H” パルス幅	65	DC	55	DC	40	DC	20	DC	ns
t_{wCl}	クロック “L” パルス幅	65	DC	55	DC	40	DC	20	DC	ns
t_{fC}	クロック立ち下がり時間		20		10		10		10	ns
t_{rC}	クロック立ち上がり時間		20		10		10		10	ns

記号	項目	CMOS 版 Z80 共通 最小	CMOS 版 Z80 共通 最大	単位
C_{CLOCK}	クロック入力端子静電容量		10	pF
$V_{IL}C$	クロック入力 “L” 電圧	−0.3	0.45	V
$V_{IH}C$	クロック入力 “H” 電圧	V_{CC}−0.6	V_{CC}+0.3	V

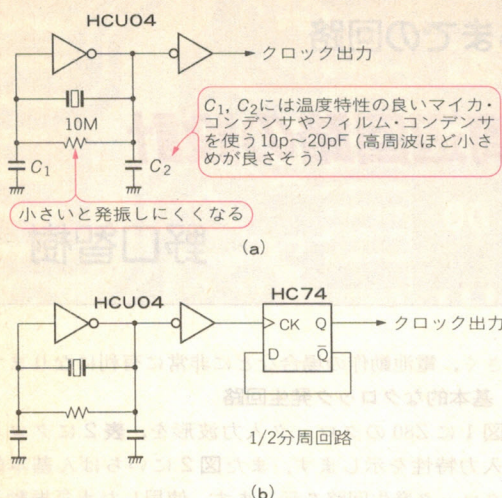
〈図 2〉 基本的なクロック発生回路

(a)

(b)

〈図 3〉 実用的な水晶発振回路

(a) インバータにTTL ICを使った水晶発振回路

(b) インバータにCMOS ICを使った水晶発振回路

〈図 4〉 実装上の注意点

をみると出力にプルアップ抵抗がついています．これは表 2 をみるとわかるように，クロック入力のHレベルの電圧が $V_{CC}-0.6$ V であるためです．しかしこうするとクロックの立ち上がり，立ち下がり時間などに影響が出るため，本当は好ましくありません．

CMOS のインバータではプルアップの必要はありません．

● **部品や実装上の注意点**

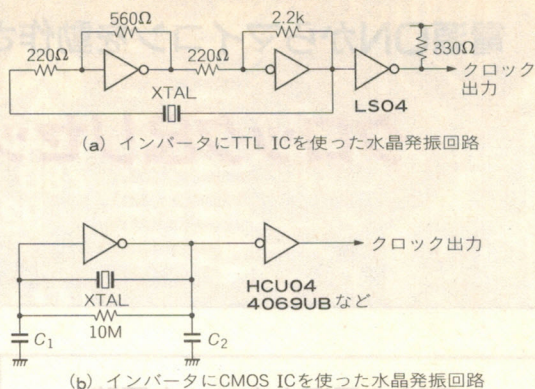
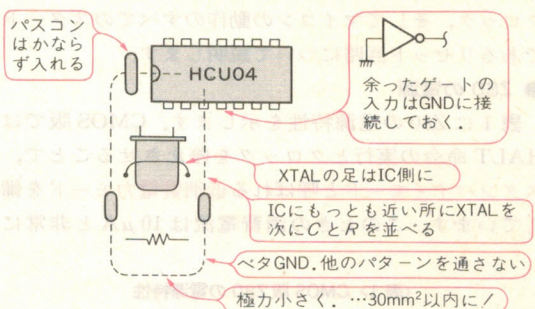
回路中のコンデンサには温度特性のよいマイカ・コンデンサやフィルム・コンデンサを使います．容量は 10 p～20 pF 程度ですが，クロック周波数が高い場合は小さめの容量がよいようです．

マイコンはディジタル回路ですが，このクロック発生回路はアナログ動作といってよいので，プリント基板化のときなどはパターン配置などに注意が必要です．

パターンは小さめにして可能な限りインバータのすぐ隣に，また水晶振動子の付近はベタ・アースで他の信号などのパターンを引き回さないようにします(図 4)．

また発振回路に使用するインバータの数は 2 個程度なので，汎用ロジック IC を使うと必ずゲートが余ります．このゲートも何かに使いたくなるでしょうが，できれば何にも使用せずに入力をグラウンドに落としたほうがよいでしょう．余ったゲートを使った場合，そのゲートのスイッチング動作が，発振周波数の精度に影響を与えます．

水晶振動子は熱に弱いので部品取り付けの順番に注意します．

またこの回路で発生できるクロック周波数は，最高十数 MHz 程度と考えてください．20 MHz 以上のクロック周波数が必要な場合は，専用のクロック・ジェネレータを使うのが賢明です．

● **専用クロック・ジェネレータを使った回路**

最近では水晶発振子と分周器を内蔵した専用のクロック・ジェネレータ IC が多数発売されています．図 2 で使用した水晶発振子とインバータによる発振回路より信頼性があり，クロックの精度も非常に高いのが特徴です．

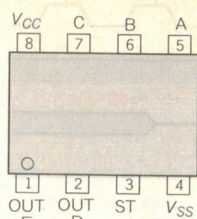
クロック・ジェネレータには EXO3 シリーズ(キンセキ)がよく使われます．これは指定クロックの発振のほかに，これを分周したクロックも出力できます．図 5 にピン配置と分周比の設定を示します．また表 3 に EXO3 が対応している周波数の一覧を示します．

● **クロック周波数の選び方**

クロック回路の設計についてはわかりましたが，では具体的にクロック周波数はどう選択すればよいのでしょう．

当然，使用する Z80 の最高クロックより高速なクロックは使用できません．4 MHz 版の Z80 なら最高 4 MHz です．

この最高周波数を越えなければ，基本的にどんな周波数でもかまいません．4 MHz 版だからといって必ず 4 MHz のクロックで使わなければならない規則はどこにもありません．また CMOS 版であれば最低クロックは DC，つまり完全にクロックを停止すること

〈図5〉 EX03のピン配置と分周比

(a) ピン配置

入力				出力	
選択			ST	F(原振周波数)	D(分周波)
C	B	A			
X	X	X	L	L	L
L	L	L	H	f_o	$f_o/2$
L	L	H	H	f_o	$f_o/2^2$
L	H	L	H	f_o	$f_o/2^3$
L	H	H	H	f_o	$f_o/2^4$
H	L	L	H	f_o	$f_o/2^5$
H	L	H	H	f_o	$f_o/2^6$
H	H	L	H	f_o	$f_o/2^7$
H	H	H	H	f_o	$f_o/2^8$

H：ハイ・レベル,
L：ロー・レベル,
X：Don't care

(b) 分周比

も可能です．

NMOS版では，CPUの内部動作が人間が自転車に乗って走っている状態のような，ダイナミック動作で動いているため，常にクロックを入力しないと動作できません．つまり最低クロック周波数というのが決まっているので，これより遅い周波数は選択できません．

NOP命令1個2個でクロック時間を計算する必要があるなら，4 MHzや10 MHzといった切りのよい周波数のほうが実行時間を計算しやすいでしょう．

また，このCPUのクロックをSIOの通信用の送受信クロックにも使うような設計も多くみられます．この場合はその通信用のクロックを分周して作り出すのに都合のよい周波数を選択すべきです．

表4に通信用クロックの分周に最適なマスタ・クロック周波数を示します．6 MHz付近では6.144 MHzと，6 MHzを少し越えていますが，これはもともと6.144 MHzでの使用を意識していて，6 MHz版のZ80は最高動作クロックが6.17 MHzになっているので大丈夫です．

〈表4〉 通信用クロックの分周に最適なクロック周波数の例

CPUクロック	システム・クロック
5MHz付近	4.915MHz
6MHz付近	6.144MHz
8MHz付近	7.987MHz
10MHz付近	9.83MHz
16MHz付近	15.974MHz
20MHz付近	19.66MHz

Z80のリセット回路

● 基本的なリセット回路

図6にZ80のリセット・クロック入力波形のようすと，**表5**にリセット・サイクルを示します．これをみるとわかるように，Z80は入力されるクロックで3クロック分の間，リセット端子をLレベルにしなければなりません．

図7に抵抗とコンデンサによるもっとも基本的なリセット回路と，各部の波形の変化のようすを示します．

リセット時間はCとRによる時定数で決まります．この回路では，

$$リセット時間 = 10\times10^3\times10\times10^{-6}\times0.66 = 0.066\ [s]$$

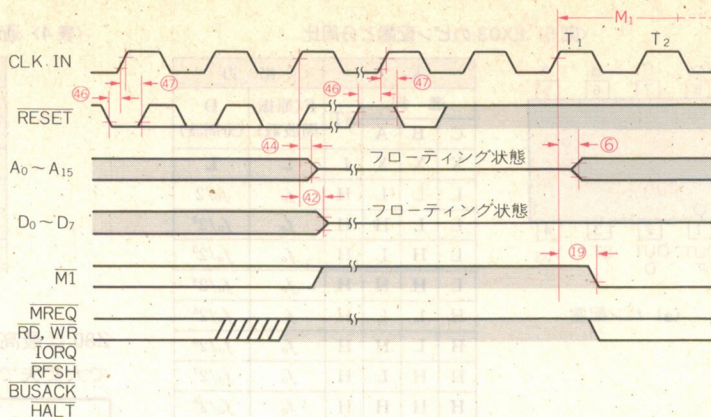
となります．

この時間がZ80CPUをリセットするために必要な，3クロック分の時間かどうかはもはや計算する必要もありませんね．1 MHz動作のZ80でも十分すぎるほどの時間です．

また間にシュミット・トリガ入力のインバータが2

〈表3〉
EX03の周波数

原発振	分周出力							
$1/2^0$	$1/2$	$1/2^2$	$1/2^3$	$1/2^4$	$1/2^5$	$1/2^6$	$1/2^7$	$1/2^8$
MHz				kHz				
12	6	3	1.5	750	375	187.5	93.75	46.875
12.288	6.144	3.072	1.536	768	384	192	96	48
12.8	6.4	3.2	1.6	800	400	200	100	50
14.31818	7.15909	3.579545	1.789772	894.88	447.44	223.72	111.86	55.930
14.7456	7.3728	3.6864	1.8432	921.6	460.8	230.4	115.2	57.6
15.9744	7.9872	3.9936	1.9968	998.4	499.2	249.6	124.8	62.4
16	8	4	2	1000	500	250	125	62.5
16.384	8.192	4.096	2.048	1024	512	256	128	64
17.73447	8.867238	4.433619	2.216809	1108.40	544.20	277.10	138.55	69.275
18.432	9.216	4.608	2.304	1152	576	288	144	72
19.6608	9.8304	4.9152	2.4576	1228.8	614.4	307.2	153.6	76.8
20	10	5	2.5	1250	625	312.5	156.25	78.125
24	12	6	3	1500	750	375	187.5	93.75

〈図6〉
Z80のリセット・サイクルのタイミング

〈表5〉 Z80のリセット・サイクル（単位 ns）

番号	記号	項目	6 MHz 版		8 MHz 版		10 MHz 版		20 MHz 版	
			最小	最大	最小	最大	最小	最大	最小	最大
6	$t_{dCr(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
19	$t_{dCr(M1f)}$	クロック立ち上がりから$\overline{\mathrm{M1}}$＝“L”になるまでの遅延		80		70		65		45
42	$t_{dCr(DZ)}$	クロック立ち上がりからデータ・バス・フロート状態までの遅延		80		70		65		40
44	$t_{dCr(AZ)}$	クロック立ち上がりからアドレス・バス・フロート状態までの遅延		80		70		75		40
46	$T_{sRESET(Cr)}$	クロック立ち上がりに対する$\overline{\mathrm{RESET}}$セットアップ時間	60		45		40		15	
47	$T_{hRESET(Cr)}$	クロック立ち上がりから$\overline{\mathrm{RESET}}$ホールド時間	10		10		10		10	

段入っているのは，Z80のリセット入力端子にヒステリシス特性がないためです．図のような徐々に電圧が上昇する信号を直接リセット端子には入力できません．

また場合によっては正論理でリセットをかける周辺LSIを使うかもしれません．このときは図のように1段目と2段目の間からリセット信号を引くとよいでしょう．

● **リセット時間の注意点**

Z80は確かに3クロック分の時間でリセットをかけられますが，このZ80のクロックに，専用のクロック・ジェネレータICを使う場合は注意が必要です．

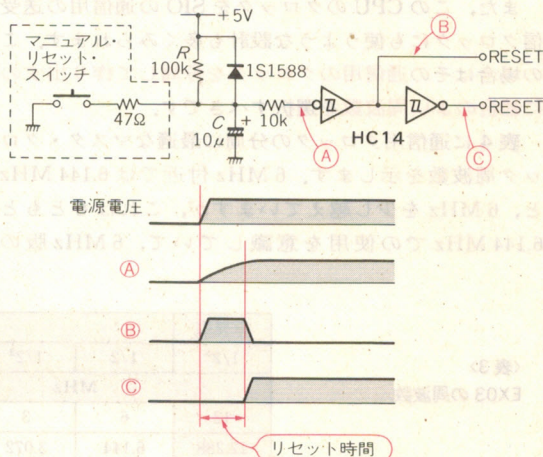
クロック・ジェネレータの中には，電源投入後数百msものクロック発振準備時間を必要とするものもあります．これらの準備時間も考えて，リセット時間を決めなければなりません．

● ***CR*によるリセット回路の問題点**

この*CR*によるリセット回路は基本的で簡単な回路なので多用されていますが，実は問題点もあります．

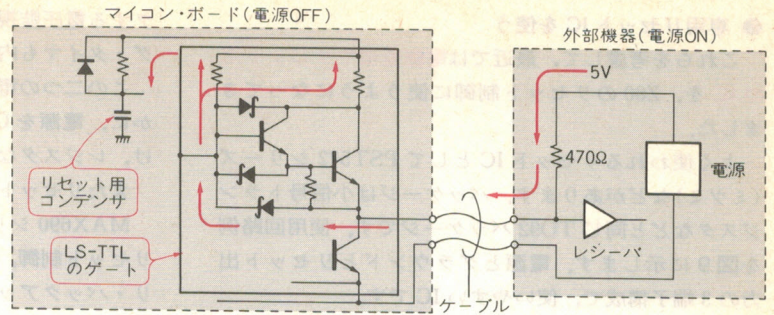
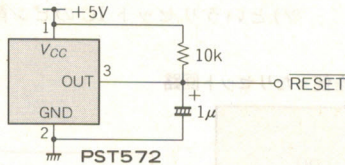
例えば，このリセット回路を搭載したCPUカードで，何らかの外部機器をコントロールするとします．電源投入の順序を外部機器から先に入れた場合，インターフェース部分のドライバに使用したICの内部構造によっては，図8のように電流がCPU側にも流れ，リセット用のコンデンサが充電されてしまいます．

このようにコンデンサが充電されたままCPU側の

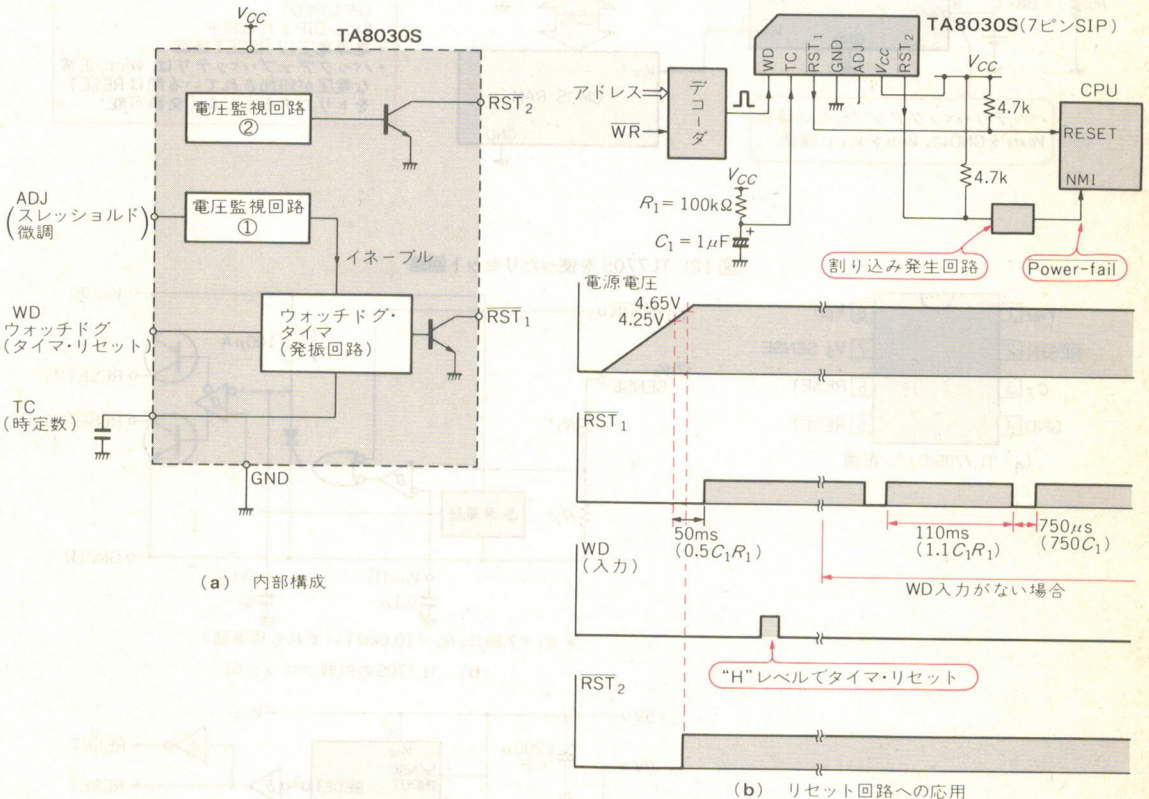
〈図7〉 *CR*によるリセット回路

電源を入れると，すでにコンデンサには電荷がたまっているわけですから，時定数の時間よりはるかに短い時間で充電が完了してしまい，Lレベルの時間が短くなってしまいます．これではCPUを正常にリセットできなくなってしまいます．

また電源電圧の立ち上がりがゆっくりな場合も正常にリセットできません．

さらに電源装置に出力電圧の不安定なものを使って

〈図 8〉
抵抗とコンデンサによるリセット回路の不都合の例

〈図 9〉
PST572 を使ったリセット回路

〈図 10〉 **TA8030S を使ったリセット回路**

(a) 内部構成

(b) リセット回路への応用

いる場合，CMOS の Z80 の場合 4.5 V が最低動作電圧ですから，たとえ瞬間的にでもこの電圧を下回ったら CPU の内部動作は保証できませんから，リセットをかけなければなりません．

しかし *CR* による回路では，すでに説明したように波形整形のためにシュミット・トリガ入力のインバータを使います．

このヒステリシス特性のために，一度 H レベルになった信号は約 1 V 程度まで電圧が下がらないと L レベルになってくれません．

● **専用リセットICを使う**

これらを考慮して，最近では電源監視やリセット用のICを，Z80のリセット制御に使うようになってきました．

よく使われるリセットICとしてPST572シリーズ(ミツミ)などがあります．パッケージは小信号トランジスタなどと同じTO92パッケージです．使用回路例を図9に示します．電源とグラウンドとリセット出力の3端子構成で，使いやすいICです．

またTA8030S(東芝)を使った回路例を図10に示します．このICは装置全体にリセットをかける目的の電圧監視回路と，電圧が低下してきたことをチェックする電圧監視回路をもっています．またウォッチドグ・タイマも内蔵しています．

この二つの電圧検出を応用するとリセット制御のほかに，電源をOFFする瞬間にCPUに割り込みをかけ，レジスタなどの退避や保存をすることができます．

またリセット時間は外付けの*CR*で決められます．

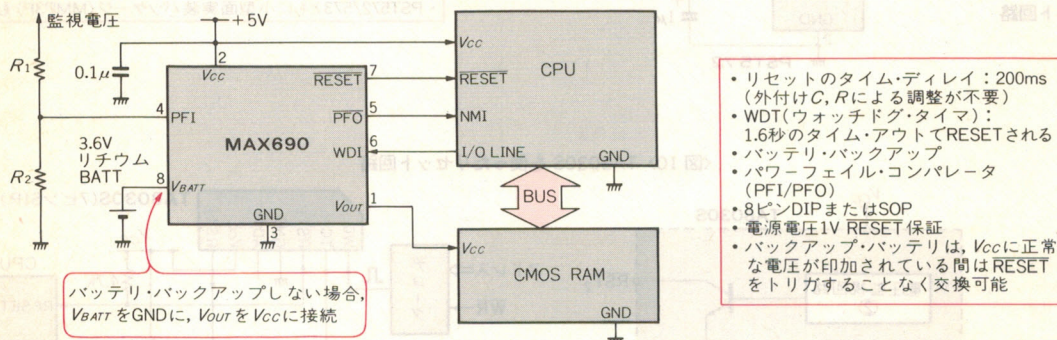
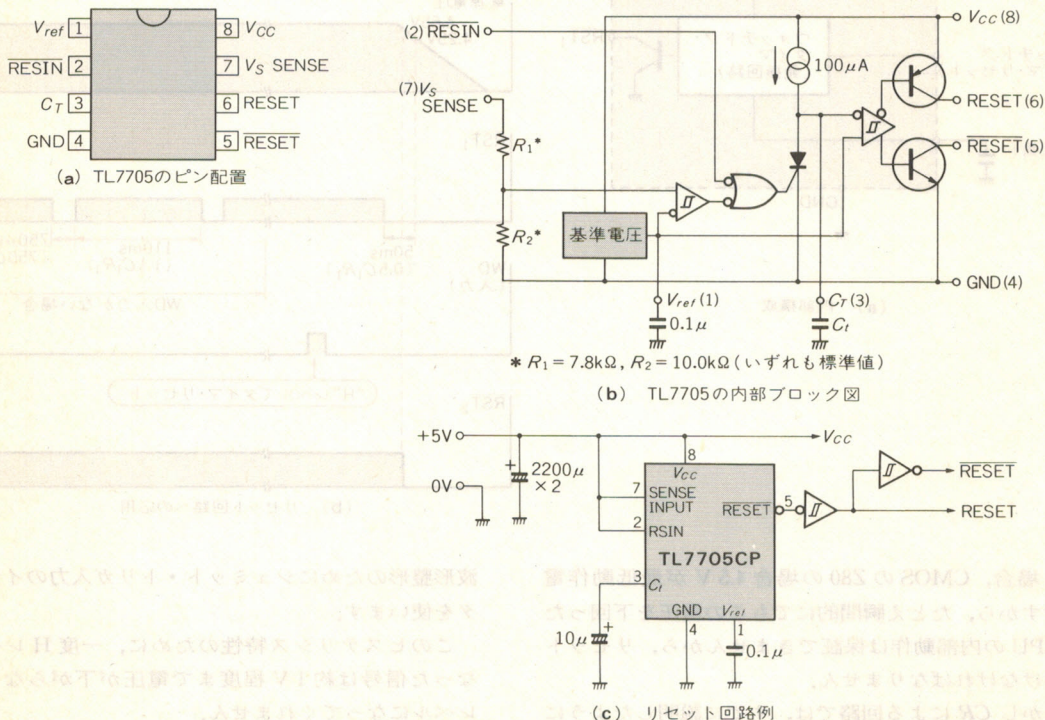
MAX690シリーズ(マキシム)も高機能なICです．リセット制御，ウォッチドグ・タイマ，そしてバッテリ・バックアップの制御機能をもっている点が特徴的です．図11に回路例を示します．

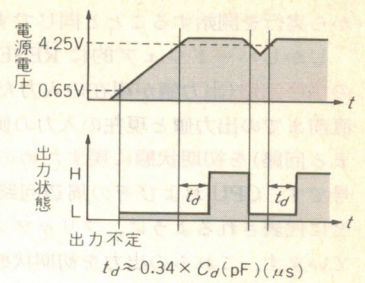
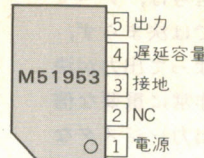
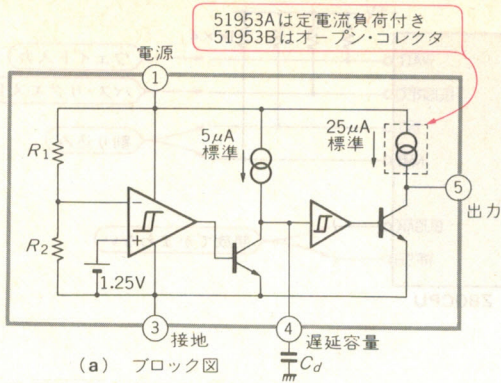
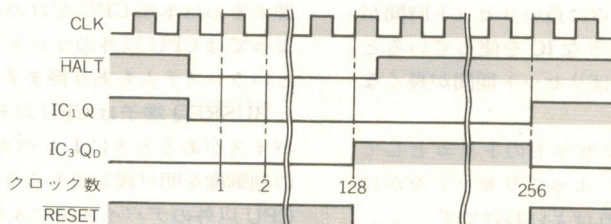
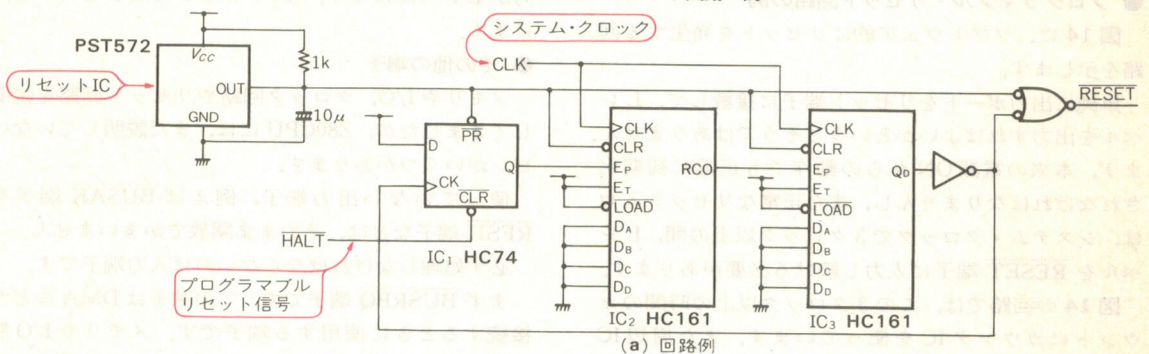
ほかに図12にTL7705(テキサス・インスツルメンツ)というリセットICのピン配置やブロック図を，最

〈図11〉MAX690を使ったリセット回路

〈図12〉TL7705を使ったリセット回路

(a) TL7705のピン配置

(b) TL7705の内部ブロック図

(c) リセット回路例

〈図13〉M51953Bの内部構造とピン配置

〈図14〉プログラマブル・リセット回路の例

後にM51953(三菱電機)のピン配置やブロック図を図13にします．

● Z80のリセットとは

ここまでZ80のリセットは，3クロック分の時間リセット入力端子をLレベルにすると説明してきましたが，もっと正確にいうと，「リセット入力端子をLレベルにして，クロック入力端子に3クロック以上クロックを入れるとリセットがかけられる」といい直しましょう．

つまりクロックが正常に入力されていない状態では，いくら数百ms，いや1秒以上リセット入力端子をLレベルにしたところで，Z80はまったくリセットされないことに注意してください．

● プログラマブル・リセットとは

メモリのパリティ・エラーなど，システムで何らかのエラーが検出された場合，その後の処理をどうするかは，システム・デザイン的な問題なので，その機器の用途や動作環境，要求される信頼性などから検討する必要があります．そのあたりの解説は，他のマイコン・システム設計などの書籍を参考にしてください．

ここでは，エラーを検出した後の処理や，リセット・スタートと同様にもう一度初めからシステムを動作させたい場合など，ソフトウェアから自分自身をリセットしたいときの方法について説明します．

リセットをソフトウェア的な面だけからみると，割り込みモードが0および割り込み受け付け禁止状態で，

JP 0000H や，RST 00H など，アドレス0000h 番地から実行を開始することと同じです．

しかしハードウェア的に $\overline{\text{RESET}}$ 信号は，すべての順序回路(出力値が現在の入力だけでは決まらず，直前までの出力値と現在の入力の値によって出力が決まる回路)を初期状態に戻すための，非常に重要な信号です．CPU およびその周辺回路は出力レジスタなどに代表されるように，フリップフロップが多用されています．これらの出力を初期状態に戻すための信号として，$\overline{\text{RESET}}$ 信号が使われるわけです．

よって，$\overline{\text{RESET}}$ 信号によって出力レジスタなどがハードウェア的に初期化される回路が使われているシステムの場合，たんに 0000h 番地から実行を再開しても，リセット・スタートと同様な動作とはなりません．

● プログラマブル・リセット回路の例

図 14 に，ソフトウェア的にリセットを発生する回路を示します．

単純に出力ポートをリセット端子に接続して，L レベルを出力すればよいかというとそうではありません．まず，本来の電源 ON からの動作でも正常に初期化されなければなりませんし，また正常なリセット入力は，システム・クロックで 3 クロック以上の間，L レベルを $\overline{\text{RESET}}$ 端子に入力し続ける必要があります．

図 14 の回路では，この 3 クロック以上の時間のカウントにカウンタ IC を使っています．また周辺 IC によっては 3 クロックよりさらに長いリセット時間が必要な物があります．このような IC を使っているときは，カウント値を多くすればリセット期間が長くなります．

図ではプログラマブル・リセットのトリガとして HALT 信号を使っています．よってリセットをかけるには HALT 命令を実行すればよいわけです．

リセット IC によるリセット出力をハードウェア・リセット，図 14 のカウンタ出力によるリセット出力をソフトウェア・リセットと呼びます．CPU やプログラム的にもリセットをかけたいデバイスのリセット入力には，この二つのリセット信号を OR した信号を入力し，両方でリセットがかかるようにします．

ポート出力によってリセットをかけたい場合は，$\overline{\text{HALT}}$ の替わりにそのポートの任意のビットを接続します．ただし，その出力ポートはハードウェア・リセット時に H レベルで初期化される必要があります．さもないと，ハードウェア・リセットとソフトウェア・リセットの両方がかかりっぱなしという状態になりかねません．

〈図 15〉その他の信号線の処理

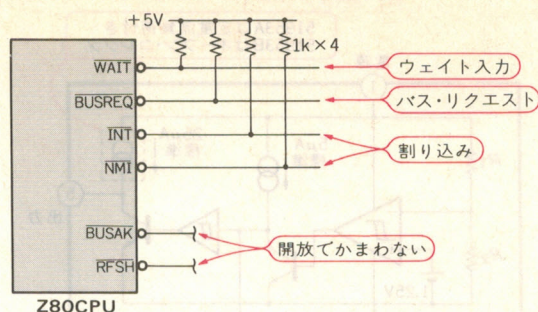



図 14 の回路では HALT 命令実行後，$\overline{\text{HALT}}$ 端子が L レベルになって 128 クロック目から $\overline{\text{RESET}}$ 出力が L レベルになり，128 クロックの間リセットをかけます．

● その他の端子

メモリや I/O，クロック回路やリセット回路を説明してきましたが，Z80CPU には，まだ説明していないピンがいくつかあります．

使っていない出力端子，例えば $\overline{\text{BUSAK}}$ 端子や $\overline{\text{RFSH}}$ 端子などは，そのまま開放でかまいません．

必ず処理しなければならないのは入力端子です．

まず $\overline{\text{BUSREQ}}$ 端子です．この端子は DMA などを接続するときに使用する端子です．メモリや I/O 制御するのは本来 CPU だけのはずです．しかし場合によっては CPU 以外のコントローラがバスを制御するというシステムもあり得ます．

$\overline{\text{BUSREQ}}$ 端子は CPU 以外にバスを制御したいデバイスがあるときに L レベルを入力し，CPU にバスの制御権を明け渡してもらうための要求信号線です．CPU 以外のデバイスがバスを制御することがない場合は電源に接続してかまいませんが，一般的にはプルアップ抵抗をつけて H レベルにしておきます．

$\overline{\text{WAIT}}$ 端子は第 4 章のコラムで少し触れたように，アクセス速度が遅いデバイスが CPU に待ったをかけるための信号線です．これも普通はプルアップしておきます．

$\overline{\text{INT}}$ と $\overline{\text{NMI}}$ は，次の割り込みの章で徹底的に解説します．これらの端子も一般的にはプルアップしておきます．

以上をまとめて，もっとも一般的な各信号端子の処理回路例を図 15 に示します．

（トランジスタ技術 1994 年 6 月号に加筆，修正）

第6章

割り込みの受け付けから割り込み処理の開始まで

Z80の割り込みシステムの動作

末木 豊/野口智樹

割り込み処理とは何か

● 割り込みとは

人間の日常生活でも，何かをしているときに急に別の何かをしなければならず，そちらの処理をしてからもとに戻ることがあります．例えば，仕事をしているときに電話がかかると，話が終わったらまた仕事に戻ります．これが割り込みです．

ではマイコンにおける割り込みとは何なのでしょう．この章では割り込み処理の考え方と，Z80の割り込み動作について解説します．

● マイコンにおける割り込み処理

まず図1(a)に外部からデータを読み込むいちばん基本的なプログラムの例を示します．これはI/OポートSTATUSのデータがLレベルになったら，I/OポートDATAからデータを読み込み，処理をしてから，ふたたび同じ動作を繰り返すものです．このように，準備ができているかどうか状態を調べて，その状態が変化したときに対応する処理を実行するやりかたをポーリングと呼びます．

しかしもしここで，STATUSが長い間Lレベルにならなかったら，CPUはその間じゅうループしていることになります．これではせっかくのCPUの処理能力をむだにしていることになります．

いまここで処理すべき事柄が，このデータの読み込みしかないのであればこれでも問題はありませんが，他にも処理すべき事柄があるときは，図1(b)のよう

(a) ポーリングとは

(b) 割り込みを使うと

〈図1〉
ポーリングと割り込み

〈図2〉 ポーリングでは難しい処理

(a) システムの処理内容

メイン・ルーチン

```
MAIN: CALL STOP
      処理Ⓐ
      CALL STOP
      処理Ⓑ
      CALL STOP
      処理Ⓒ
      JP   MAIN

STOP: PUSH AF
      IN   A, (STOP_SW)
      OR   A
      JP   Z, STOP_ON
      POP  AF
      RET

STOP_ON: 非常停止処理
```

いたるところで非常停止スイッチが押されているかどうかをチェックしなければならない

処理の途中でスイッチが押されてもプログラムはかわらない➡すぐに止まらない！

非常停止スイッチが押されているかチェックするサブルーチン

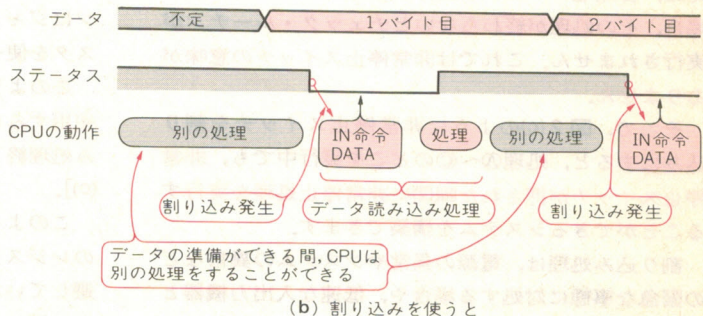
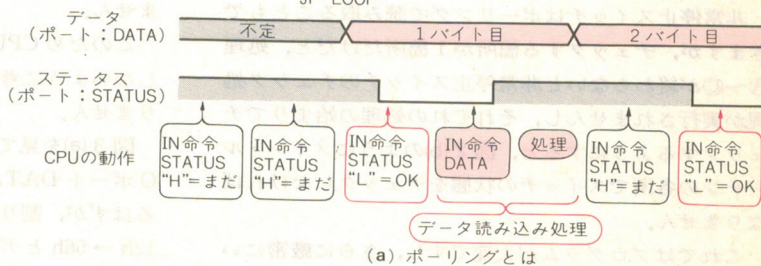
(b) ポーリングによる状態のチェック

メイン・ルーチン

```
MAIN: 処理Ⓐ
      処理Ⓑ
      処理Ⓒ
      JP MAIN

STOP_ON: 非常停止処理
```

非常停止スイッチが押された！

割り込みが発生して非常停止処理ルーチンへジャンプする

(c) 割り込みを使用すると

にするとどうでしょう．

これはステータス信号の立ち下がりをきっかけにしてCPUに信号を送り，その瞬間だけデータの読み込み処理をしてもらいます．処理が終わったらもとの処理に戻します．これが，割り込み処理の一例です．

● ポーリングでは難しい処理

次に図2をみてください．これは処理Ⓐ～Ⓒを繰り返し実行しながら，非常停止するというマイコン・システムの例です．

非常停止スイッチはポーリングで読み取ることもできますが，チェックする箇所が1箇所だけだと，処理Ⓐ～Ⓒが終わらないと非常停止スイッチのチェック処理が実行されませんし，それぞれの処理の始まりでチェックするようにすると，図2(b)のようにメイン・ルーチンの各所でスイッチの状態をチェックしなければなりません．

これではプログラムが大変ですし，さらに厳密にいえば，各処理ルーチンの実行中にスイッチが押された場合，その処理が終わらないとチェック・ルーチンが実行されません．これでは非常停止スイッチの意味がありません．

そこで，図2(c)のように非常停止スイッチを割り込みにすると，処理Ⓐ～Ⓒのどこを実行中でも，非常停止スイッチが押された瞬間に非常停止処理を実行することができるシステムを構築できます．

割り込み処理は，電源の異常やシステムの異常などの緊急な事態に対処する場合や，低速な入出力機器とのデータ転送を，待ち時間なしに効率よく行う場合，またいつ発生するかわからない事柄を，別の処理をしながら待つときなどに使います．

このように割り込みとは，CPUのもつ処理能力をむだなく使用することのできる，マイコン・システムにおける重要な概念です．

● 割り込み処理のプログラム

すでに説明したように割り込み処理は，それまでの実行中のプログラムの流れを突然変え，用が済んだらふたたび何事もなかったかのように戻らなければなりません．

このためCPUのレジスタやメモリを不用意に破壊しないように考慮して，プログラムを組まなければなりません．

図3(a)を見てください．割り込みがないときは，I/OポートDATA0に12h→34hとデータが出力されるはずが，割り込みの発生するタイミグによっては，12h→56hとデータが出力されてしまいます．これはAレジスタに34hを代入したとたん割り込みルーチンにジャンプし，さらに割り込みルーチンでAレジスタを使っているためです．

このようなことのないよう，割り込みルーチンでは使用するレジスタやフラグはすべて待避して，割り込み処理終了後に復帰してから戻るようにします［図3(c)］．

このように基本的には割り込み処理の先頭ですべてのレジスタをスタックに退避し，処理が終わったら退避していたレジスタを戻して実行途中だったプログラムに戻ります．

〈図3〉
割り込みルーチンでレジスタを保存しておく

(a) 割り込みがないとき

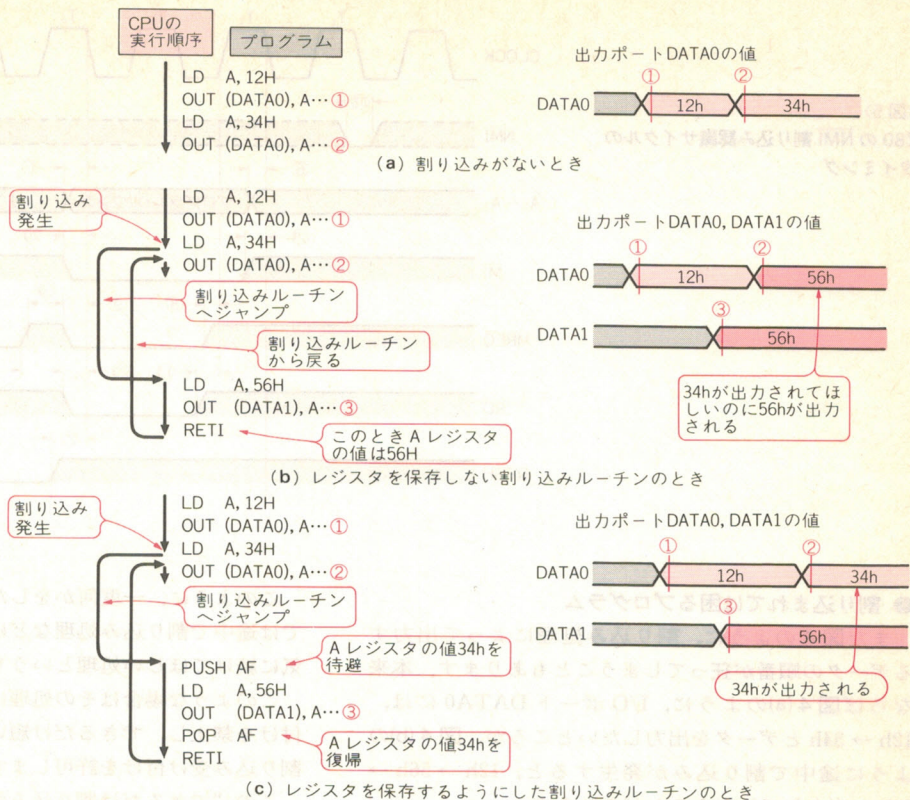
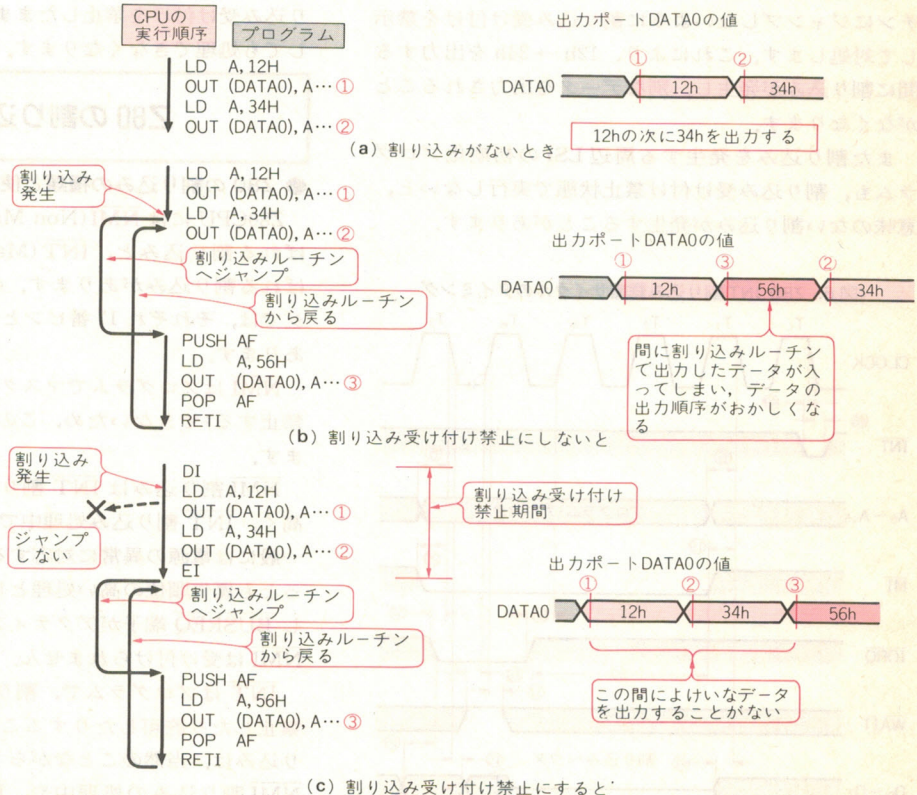
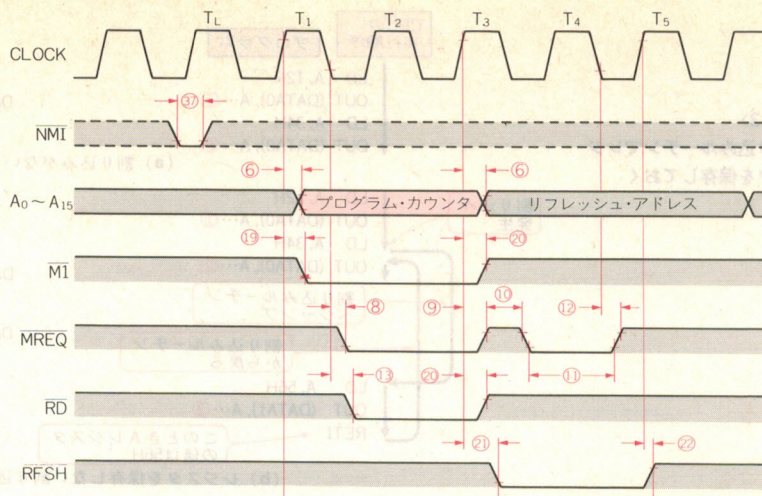
(b) レジスタを保存しない割り込みルーチンのとき

(c) レジスタを保存するようにした割り込みルーチンのとき

〈図4〉
割り込まれたくないプログラムもある

(a) 割り込みがないとき

(b) 割り込み受け付け禁止にしないと

(c) 割り込み受け付け禁止にすると

〈図5〉
Z80のNMI割り込み認識サイクルのタイミング

● 割り込まれては困るプログラム

また図4のように，割り込み処理によって出力するデータの順番が狂ってしまうこともあります．本来ならば図4(a)のように，I/OポートDATA0には，12h → 34hとデータを出力したいところが，図4(b)のように途中で割り込みが発生すると，12h → 56h → 34hと出力してしまうことがあります．

この場合は図4(c)のように，途中で割り込みルーチンにジャンプしないように割り込み受け付けを禁示して対処します．これにより，12h → 34hを出力する間に割り込みが発生し，別なデータを出力されることがなくなります．

また割り込みを発生する周辺LSIの初期化プログラムも，割り込み受け付け禁止状態で実行しないと，意味のない割り込みが発生することがあります．

〈図6〉 Z80 INT割り込み認識サイクルのタイミング

このように，一度何かをしだしたら，あるところまでは途中で割り込み処理などにジャンプしないで，一気に続けてほしい処理というものがあります．

このような場合はその処理に入る前に割り込み受け付けを禁止し，できるだけ短い時間で処理をすませて割り込み受け付けを許可します．

この“できるだけ割り込み受け付け禁止状態を短くする”というのがポイントで，不必要に長い時間，割り込み受け付けを禁止したままでは，割り込みが発生しても処理できなくなります．

Z80の割り込み制御

● Z80の割り込みの種類と使い分け

Z80CPUにはNMI(Non Maskable Interrupt)と呼ばれる割り込みと，INT(Maskable Interrupt)と呼ばれる割り込みがあります．40ピンのDIPパッケージでは，それぞれ17番ピンと16番ピンに入力端子があります．

NMIはプログラムでマスク(割り込みの受け付けを禁止する)できないため，このような名前になっています．

NMI割り込みはINT割り込みよりも優先順位が高く，INT割り込み処理中でも受け付けられるので，一般には電源の異常に対応する処理などのような，もっとも優先順位の高い処理として使用されます．ただしBUSREQ端子がアクティブになっている場合には，NMIは受け付けられません．

INTはプログラムで，割り込み処理の受け付けを禁止したり許可したりすることが可能です．INT割り込みは，当然のことながらこれより優先順位の高いNMI割り込みの処理中や，BUSREQ端子がアクテ

〈表 1〉 Z80 の割り込み認識サイクル(単位：ns)

番号	記　号	項　　目	6MHz 版		8MHz 版		10MHz 版		20MHz 版	
			最小	最大	最小	最大	最小	最大	最小	最大
6	$t_{dCr(A)}$	クロック立ち上がりからの有効アドレス出力遅延		90		80		65		57
8	$t_{dCf(MREQf)}$	クロック立ち下がりからの $\overline{MREQ}$＝“L”になるまでの遅延		70		60		55		40
9	$t_{dCr(MREQr)}$	クロック立ち上がりからの $\overline{MREQ}$＝“H”になるまでの遅延		70		60		55		40
10	t_{wMREQh}	$\overline{MREQ}$＝“H”のパルス幅	65		45		30		10	
11	t_{wMREQl}	$\overline{MREQ}$＝“L”のパルス幅	132		100		75		25	
12	$t_{dCf(MREQr)}$	クロック立ち下がりから $\overline{MREQ}$＝“H”になるまでの遅延		70		60		55		40
13	$t_{dCf(RDf)}$	クロック立ち下がりから $\overline{RD}$＝“L”になるまでの遅延		80		70		65		40
15	$t_{sD(Cr)}$	クロック立ち上がりに対するデータ・セットアップ時間	30		30		25		12	
16	$t_{hD(RDr)}$	$\overline{RD}$ 立ち上がりに対するデータ・ホールド時間	0		0		0		0	
17	$t_{sWAIT(Cf)}$	クロック立ち下がりに対する $\overline{WAIT}$ 信号セットアップ時間	60		50		20		7.5	
18	$t_{hWAIT(Cf)}$	クロック立ち下がり後の $\overline{WAIT}$ ホールド時間	10		10		10		10	
19	$t_{dCr(M1f)}$	クロック立ち上がりから $\overline{M1}$＝“L”になるまでの遅延		80		70		65		45
20	$t_{dCr(M1r)}$	クロック立ち上がりから $\overline{M1}$＝“H”になるまでの遅延		80		70		65		45
21	$t_{dCr(RFSHf)}$	クロック立ち上がりから $\overline{RFSH}$＝“L”になるまでの遅延		110		95		80		60
22	$t_{dCr(RFSHr)}$	クロック立ち上がりから $\overline{RFSH}$＝“H”になるまでの遅延		100		85		80		60
37	t_{wNMI}	$\overline{NMI}$ パルス幅	60		60		60		60	
42	$t_{dCr(DZ)}$	クロック立ち上がりからデータ・バス・フロート状態までの遅延		80		70		65		40
48	$t_{sINTf(Cr)}$	クロック立ち上がりに対する $\overline{INT}$ セットアップ時間	70		55		50		15	
49	$t_{sINTr(Cr)}$	クロック立ち上がり後の $\overline{INT}$ ホールド時間	10		10		10		10	
50	$t_{dM1f(IORQf)}$	$\overline{IORQ}$ 立ち上がりに先き立つ $\overline{M1}$ 出力(L)の確定時間	359		270		220		100	
51	$t_{dCf(IORQf)}$	クロック立ち下がりから $\overline{IORQ}$＝“L”になるまでの遅延		70		60		55		45
52	$t_{dCf(IORQr)}$	クロック立ち下がりから $\overline{IORQ}$＝“H”になるまでの遅延		70		60		55		45

ィブになっている場合には受け付けられません．

また $\overline{INT}$ 割り込みにはモード 0〜2 の三つのモードがあります．モード 0 は Z80 の原型ともなった 8080A という CPU と互換性をもたせるためのモードです．またモード 1 も動作原理はモード 0 と同じです．

モード 2 が Z80 本来の割り込みモードで，このモード 2 と Z80 周辺 LSI を組み合わせることによって，他の CPU 系には見られない独特な(?)割り込みシステムを構成することができます．

図 5 と図 6，そして**表 1** に，Z80 の割り込み認識サイクルを示します．

● 割り込み制御に関係のある命令

Z80 の数ある命令のうち，割り込みの制御を行う命令が 10 個ほどあります．それぞれを簡単に説明すると，**表 2** のようになります．

表中の P/V は，パリティ・フラグのことです．また IFF_1は，$\overline{INT}$ 割り込み処理の受け付け許可/禁止を制御する CPU 内部にあるフリップフロップで，IFF_2は，$\overline{NMI}$ 発生時に $\overline{INT}$ 割り込み受け付けが許可されているか禁止されているかを保存しておくフリップフロップです．

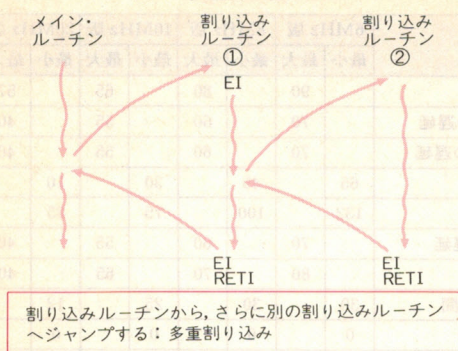
IM 0〜IM 2 は割り込みモードの設定です．またマスク可能な割り込みですから，割り込み受け付けを禁止する命令である DI，許可する命令である EI があります．

〈表 2〉 Z80 の割り込み制御に関係のある命令

命　令	動　　作
IM 0	$\overline{INT}$ 割り込みをモード 0 に設定する．
IM 1	$\overline{INT}$ 割り込みをモード 1 に設定する．
IM 2	$\overline{INT}$ 割り込みをモード 2 に設定する．
EI	$\overline{INT}$ 割り込み受け付けを許可する． $IFF_2 \leftarrow 1$，$IFF_1 \leftarrow 1$
DI	$\overline{INT}$ 割り込み受け付けを禁止する． $IFF_2 \leftarrow 0$，$IFF_1 \leftarrow 0$
RETI	$\overline{INT}$ 割り込み処理からの復帰．
RETN	$\overline{NMI}$ 割り込み処理からの復帰．
LD A, I	I レジスタの内容を A レジスタに転送する． $P/V \leftarrow IFF_2$
LD I, A	A レジスタの内容を I レジスタに転送する．
LD A, R	R レジスタの内容を A レジスタに転送する． $P/V \leftarrow IFF_2$

また基本的に割り込み処理プログラムは，メイン・ルーチンとは別なプログラムで処理されます．つまりその割り込み処理ルーチンからメイン・ルーチンに戻るための，専用の RET 命令があります．$\overline{NMI}$ 処理ルーチンから戻るときに使われる RETN，$\overline{INT}$ 処理ルーチンから戻るときに使われる RETI です．

またモード 2 で使用されるインタラプト・レジスタへの値の設定は，LD I, n という直接値を代入する命令がありません．かならず A レジスタに設定してから I レジスタに代入します．

〈図7〉多重割り込みとは

ここでリフレッシュ・レジスタであるRレジスタの内容をAレジスタに読み込むLD A, Rという命令が，なぜ割り込み制御に関係する命令なのでしょうか．

その秘密は実行後のフラグの変化にあり，パリティ・フラグにIFF_2の内容がコピーされるのです．つまり現在Z80が割り込み受け付け禁止状態であるか許可状態であるかを，プログラムでチェックできるのです．

また割り込みのモードは，Z80CPU内部のIMF_aとIMF_bの二つのフリップフロップに格納されます．表3にIMF_aとIMF_bの割り込みモードとの関係を示します．リセット直後の状態は8080Aとの互換性のために，モード0になります．

以上から，現在割り込み受け付け禁止状態であるか許可状態であるかは，LD A, R命令などのフラグの変化で知ることができますが，現在の割り込みモードが何であるかをプログラムが知ることはできません．IMFの状態を読み出す命令がないからです．

● 多重割り込み

Z80には$\overline{\text{NMI}}$と$\overline{\text{INT}}$の2本の割り込み入力端子がありますが，$\overline{\text{NMI}}$は緊急事態通知用として考えると，通常使える割り込みは一つしかありません．

となると優先度の高い割り込みも優先度の低い割り込みも，同一の割り込み入力端子に接続されることになります．

もし優先度の低い割り込み処理が先に発生し，その処理が終わるまで優先度の高い割り込みが待たされてしまうのでは，何のための優先順位かわかりません．優先度の低い割り込み処理中には，優先度の高い割り込みを許可しておく必要があります．

このように割り込み処理中に，別の割り込み処理を行うことを，多割り込み処理といいます(図7)．

実際には，割り込み処理ルーチンの先頭付近で，EI命令を実行することで割り込み受け付け許可状態にします．しかし単純に割り込み受け付けを許可してしまうと，優先度の低い割り込みまで受け付けてしまいそうです．

〈表3〉IMF_a，IMF_bと割り込みモードの関係

IMF_a	IMF_b	割り込みモード
0	0	モード0
0	1	未使用
1	0	モード1
1	1	モード2

このためモード0やモード1の割り込み処理の場合は，プログラム的に優先順位を管理する必要があります．

Z80本来の割り込みモードであるモード2とZ80周辺LSIの組み合わせでは，この優先順位をハード的に決定できる(デイジィ・チェーン接続)ので，多重割り込み処理の管理が非常に簡単になります(多重割り込みやデイジィ・チェーンについては後述)．

$\overline{\text{NMI}}$割り込み

● $\overline{\text{NMI}}$割り込みとは

Z80の割り込み制御のうち比較的動作が簡単で理解しやすいのが$\overline{\text{NMI}}$割り込みです．

すでに述べたように，本来$\overline{\text{NMI}}$は電源の異常やメモリのパリティ・エラーが発生した場合などの，緊急事態をCPUに知らせるための割り込みとして使われます．

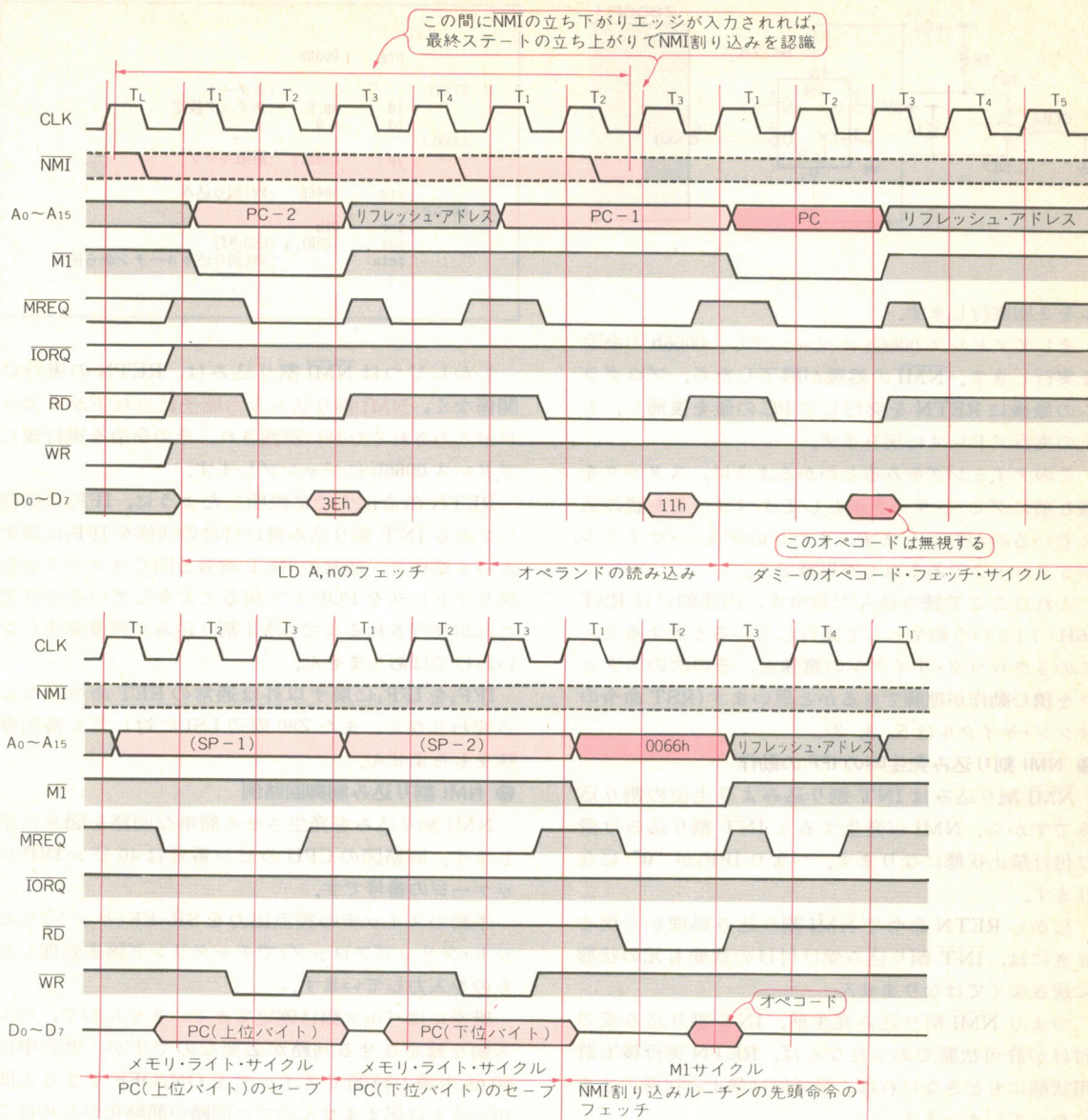
しかし，これはZ80をコンピュータのCPUとして使っていた頃の考え方で，別に必ず電源異常のチェックに使わなければならないという決まりはどこにもありません．割り込みが2本あれば間に合うシステムなら，一般的な割り込み処理として$\overline{\text{NMI}}$を使ってもかまわないでしょう．

しかし産業用機器のコントローラとして使う場合や，信頼性を少しでも向上させたいときなどは，$\overline{\text{NMI}}$をこれらの目的に使うのが正統的な使い方のようです．

$\overline{\text{NMI}}$は優先度が高いといっても，$\overline{\text{BUSREQ}}$端子がアクティブになっている場合は受け付けられません．$\overline{\text{BUSREQ}}$端子がアクティブな状態とは，CPU以外の別のコントローラが，CPUが制御しているバスの制御権を要求しているということで，例えばDMA (Direct Memory Access)コントローラなどが，メモリの読み書きをしたいことをCPUに通知している状態です．

$\overline{\text{BUSREQ}}$がアクティブになったら，CPUはそのときの命令を実行後，すみやかにバスの制御権をゆずらなくてはなりません．

よってCPUは，バスの制御権を渡した後は$\overline{\text{NMI}}$だろうがなんだろうが，制御権が返されるまでは動くことすらできません．

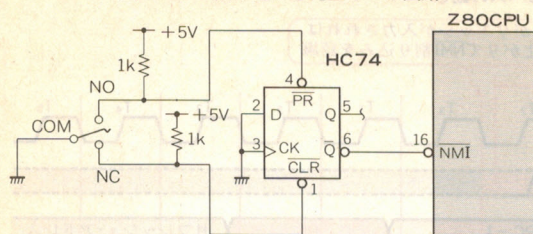
〈図8〉 $\overline{\mathrm{NMI}}$ 割り込みのバスの動き

またある意味で，あらゆるCPUにとって最上位の割り込みはリセット入力であるとも考えられます．Z80の場合，どんなバス・サイクルからでも割り込みがかかり，割り込みモードは0に，割り込み受け付けは禁止に，そして有無をいわさずアドレス0000hに実行が移されるわけですから….

● $\overline{\mathrm{NMI}}$ 割り込みのバスの動作

$\overline{\mathrm{NMI}}$ 入力は信号の立ち下がりエッジが基準となります(エッジ・トリガ)．立ち下がりエッジの後のLレベルの期間は，CPUのクロック・スピードにより違います．例えば4MHz動作のZ80で最低80ns以上なので，外部からの信号としては80ns以上のLレベルのパルスを入力すれば，$\overline{\mathrm{NMI}}$ 割り込み受け付けのCPU内部のフラグをONすることができます．

図8に $\overline{\mathrm{NMI}}$ 割り込みのバスの動作を示します．このフラグは，直前の命令のマシン・サイクルの最終ステートの立ち上がりでチェックされ，$\overline{\mathrm{NMI}}$ 割り込みが認識されます．このタイミングは $\overline{\mathrm{INT}}$ 割り込みも共通です．

そして，次に実行しようとしていたプログラム・カウンタのアドレスの命令を読み込みます．しかしここで読み込んだオペコードは，CPUには取り込まれず破棄されます．そしてPCの上位バイト，PCの下位バイトをスタックに待避すべくメモリ・ライト・サイク

〈図9〉 NMI 割り込みを発生させる回路

〈リスト1〉 NMI 割り込みで LED を反転表示するプログラム

```
        org     0000h

START:                  ;スタート
        ld      sp,0    ;スタック設定
        ld      a,0
LABEL1:
        jp      LABEL1  ;無限ループ

        org     0066h   ;NMI割り込み
NMITOP:
        xor     01h
        out     (20H),a ;LED点灯
        retn            ;NMI割り込みルーチンから戻る
```

ルを2回実行します．

そしてアドレス 0066h にジャンプし，0066h の命令を実行します．NMI の処理が終了したら，プログラムの最後に RETN を実行して PC の値を復帰し，もとの実行アドレスに戻ります．

このタイミングをみるとわかるように，スタックを積む前にダミー・サイクルとしてオペコードを読み込んでいるのがわかります．またこのダミー・サイクルが5クロックである点も不思議です．

これはここで読み込んだ命令を，内部的には RST 66H(？)という命令として実行していると考えると，この5クロック・サイクルの意味と，その次のスタックを積む動作が理解できるかと思います(RST 命令のマシン・サイクルは5，3，3)．

● NMI 割り込み発生時の IFF の動作

NMI 割り込みは INT 割り込みより上位の割り込みですから，NMI が発生すると INT 割り込みは受け付け禁止状態になります．つまり IFF_1が“0”になります．

しかし RETN 命令で NMI 割り込み処理から戻るときには，INT 割り込み受け付けの状態も元の状態に戻さなくてはなりません．

つまり NMI 割り込み発生前，INT 割り込み受け付けが許可状態であったならば，RETN 実行後も許可状態にもどさなければ，受け付け禁止の状態のままになってしまいます．

このために IFF_1と IFF_2の二つのフラグがあり，RETN 命令は IFF_2の状態を IFF_1にコピーして INT 割り込みの受け付け状態を戻します．

この動作のおかげで，NMI 処理の後でも INT 割り込みを受け付けることができます．よって NMI 割り込み処理終了は必ず RETN 命令を実行するようにします．

● RETN 命令と NMI 割り込み

もし NMI 処理プログラムが長く，NMI 処理中に再び NMI 割り込みが発生したらどうなるでしょうか．NMI 処理プログラムの最後には RETN 命令を入れますが，これが実行されるまでは NMI は再度発生しないようにも思えます．

しかしじつは NMI 割り込みは，RETN の実行に関係なく，NMI 割り込み入力端子に立ち下がりエッジが入力されるたびに認識され，その命令を実行後にアドレス 0066h にジャンプします．

RETN 命令はすでに説明したように，IFF_2に保存してある INT 割り込み受け付けの状態を IFF_1に戻すという処理と，通常の RET 命令と同じスタックから戻りアドレスを POP して戻ることをしているだけで，これが実行されるまで NMI 割り込みが再度発生しないわけではありません．

IFF_2を IFF_1に戻す以外は通常の RET 命令となんら変わりなく，また Z80 周辺 LSI に対しても特別意味をもちません．

● NMI 割り込み制御回路例

NMI 割り込みを発生させる簡単な回路を図9に示します．回路図の CPU のピン番号は 40 ピン DIP パッケージの番号です．

手動のスイッチの接点出力を SR-FF(セット/リセット・フリップフロップ)でチャタリング除去処理したものを入力しています．

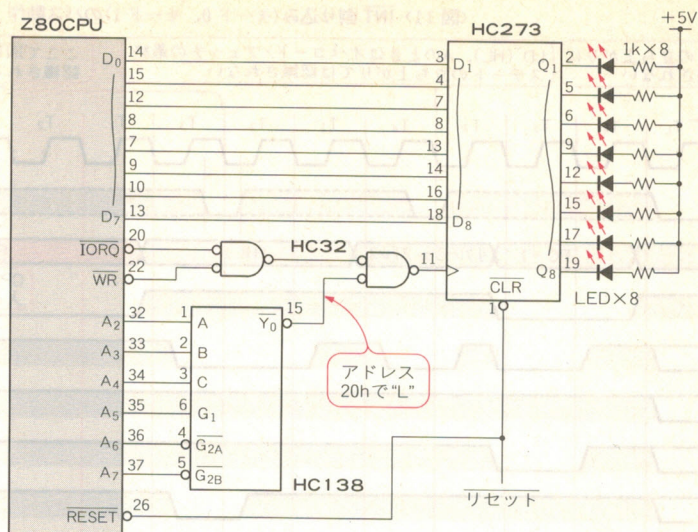
厳密にはパルス幅は確定できていませんので，パルス幅を確定させる回路が必要なのですが，世の中に 60 ns 未満の時間でスイッチを ON/OFF できる人間がいるとは思えませんので，回路の簡略化のためにここでは省略します．

また割り込み信号を入れて割り込みが認識されても，それは CPU 内部の話なので動作の確認ができません．そこで図10に示すような，簡単な出力回路を作り，NMI の動作を確認してみます．

図10の I/O 回路については第4章で解説済みなのでここでは触れません．I/O アドレスは 20h です(正確にはイメージ・アドレスが発生しているので 20 h～23 h となる)．

● NMI 割り込み処理プログラム

リスト1に NMI 割り込みで LED を反転表示するプログラムを示します(メモリについては ROM32 K バイト，RAM32 K バイトの Z80 システムを想定して

〈図10〉
LED 点灯回路

いる)．$\overline{\text{INT}}$ 割り込みがどのモードであろうとも $\overline{\text{NMI}}$ が上位の割り込みであることにかわりありません．

プログラムの動作を説明します．まずリセット直後は，アドレス0000hから実行が開始されます．まずスタック・ポインタをセットします．そして表示データを保持するAレジスタに0を代入して値をクリアしています．

そして次のアドレス0006hには，0006hにジャンプする命令を書いてあります．つまりプログラムの流れとしては，アドレス0006hへ永久にジャンプを繰り返すだけで何もしていません．

さて，ここで外部からの $\overline{\text{NMI}}$ 割り込みが発生すると，プログラム実行アドレスはハードウェア的に分岐され，アドレス0066hに実行が移ります．

Aレジスタと01hとの排他的論理和(XOR)を計算しているので，Aレジスタの最下位ビットだけが反転します．この値をI/Oアドレス20hにOUT命令で出力します．そしてRETN命令で復帰します．

このプログラムを実行すると，スイッチが押されるたびに最下位ビットのLEDがついたり消えたりします．

この例は割り込みの動作チェックのためのものなので，割り込みルーチンの先頭でレジスタの値を保存していません．しかし，実際の割り込み処理では必ずレジスタの値は保存してください．

$\overline{\text{INT}}$ 割り込み

● $\overline{\text{INT}}$ 割り込みとは

$\overline{\text{NMI}}$ 割り込みと違い，$\overline{\text{INT}}$ 割り込みは通常の割り込み処理用の入力端子です．またすでに述べたようにZ80にはモード0〜2の割り込みモードがあります．

Z80のリセット直後は，$\overline{\text{INT}}$ 信号に対する割り込み受け付けが禁止状態となっています．また割り込みのモードはモード0です．

モード1またはモード2の割り込みを使用する場合には，IM1またはIM2命令を実行する必要があります．

● $\overline{\text{INT}}$ 割り込みのしくみ

$\overline{\text{NMI}}$ 割り込みと違うところは，$\overline{\text{INT}}$ 入力は信号のLレベルが基準となることです(レベル・トリガ)．

CPUは命令の最後のステートの立ち上がりで $\overline{\text{INT}}$ 入力をサンプリングし，Lレベルかどうかをチェックします．

ここで $\overline{\text{INT}}$ 割り込みが認識されれば，次に割り込みモードによって，各モードの割り込みの応答サイクルが始まります．

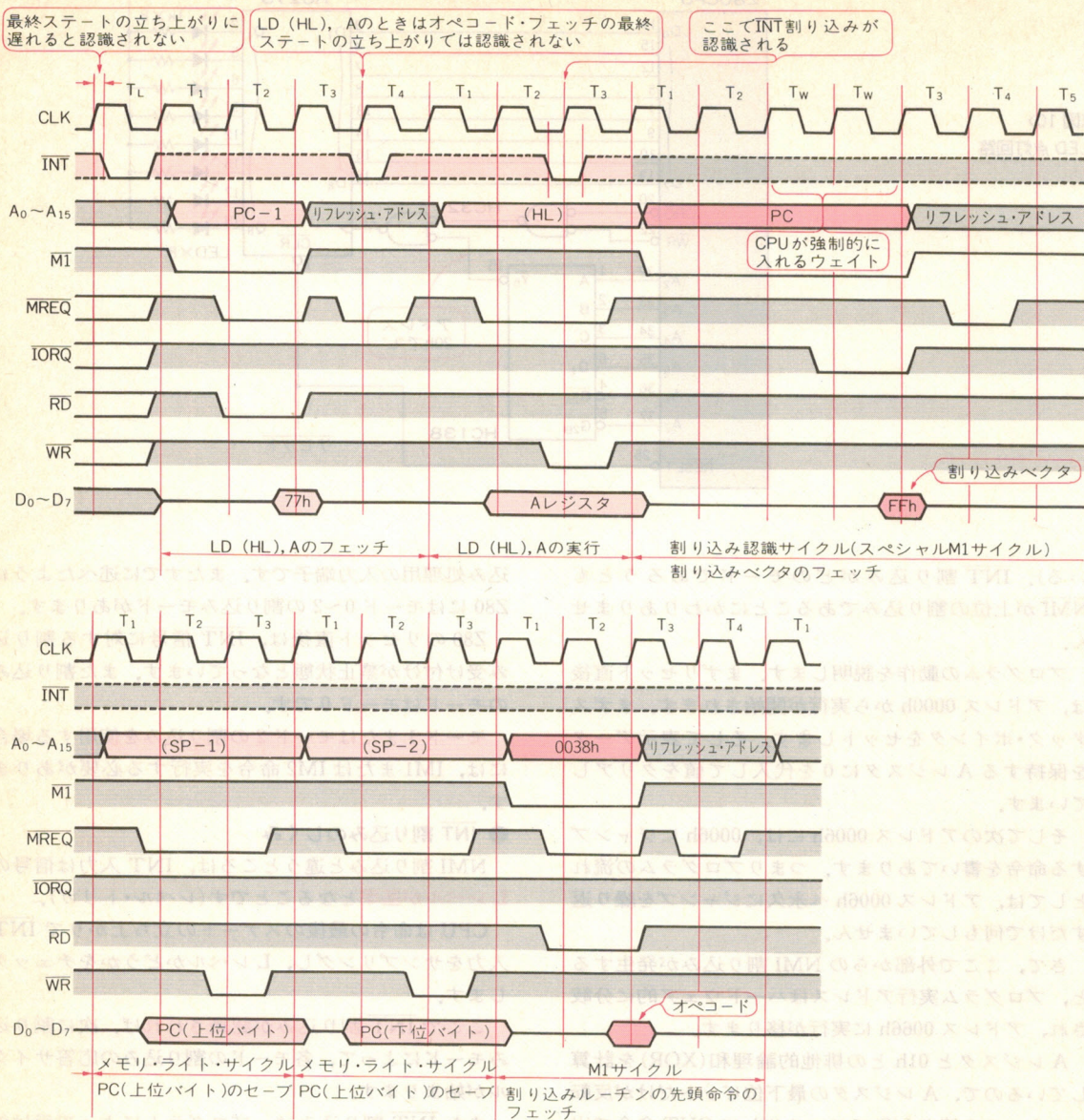
また $\overline{\text{INT}}$ 割り込みは，プログラムによって受け付けを禁止したり許可したりできます．

割り込み受け付けを許可するにはEI命令を実行します．禁止するにはDI命令を実行します．またリセット信号の入力や $\overline{\text{NMI}}$ の受け付けによっても禁止できます．

そして重要なのは，$\overline{\text{INT}}$ 割り込み自身が発生したときにも，再度 $\overline{\text{INT}}$ 割り込みが発生しないように受け付けが禁止されます．つまり一度 $\overline{\text{INT}}$ が発生すると，EI命令を実行しない限りもう一度 $\overline{\text{INT}}$ 割り込み信号が入力されても認識されません．これも $\overline{\text{NMI}}$ と違う点です．

表4に IFF_1 と IFF_2 の動作をまとめます．

それではモード0〜2をそれぞれ説明します．

〈図 11〉 $\overline{\text{INT}}$割り込み(モード 0, モード 1)のバス動作

モード0割り込みの動作

● モード0割り込みとは

モード0割り込みは8080Aと同じ動作をする割り込みです．といっても1973年に発表され，Z80や8085の出現で急速に製品寿命を閉じた8080CPUを，どれだけの人が知っているか(名前だけでなくその動作まで)はわかりませんが…．CPUチップの実物はおろか，データ・シートさえ見たこともない人がほとんどでしょう．

● モード0割り込みのバスの動作

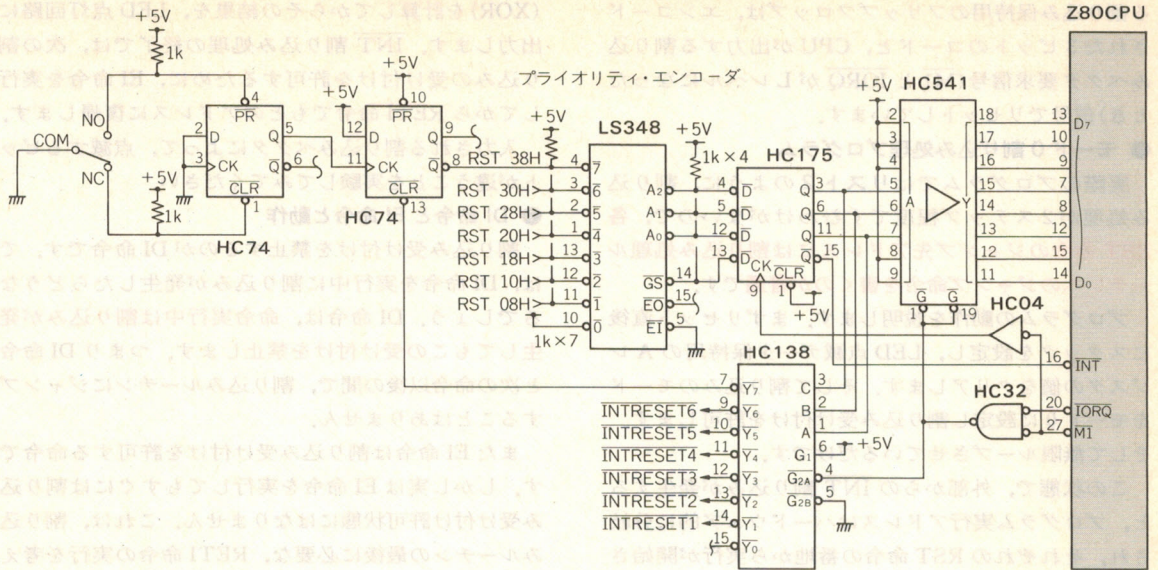
割り込を受け付けるタイミングは$\overline{\text{NMI}}$と同一ですが，割り込み応答サイクルに特徴があります．

図11にモード0のときの，$\overline{\text{INT}}$割り込みのバスの動作を示します．

$\overline{\text{INT}}$割り込み認識後，CPUは$\overline{\text{M1}}$と$\overline{\text{IORQ}}$をLレベルにして，データ・バスから割り込みベクタを読み込もうとします．

モード0のときの割り込みベクタとは，Z80が実行できる命令(オペコード)です．

この割り込みベクタとして使う命令は，Z80の命令

〈図12〉 $\overline{\text{INT}}$ 割り込み(モード0)を発生させる回路

〈表4〉 IFF_1 と IFF_2 の動作

	IFF_1	IFF_2
リセット	0	0
DI 命令	0	0
EI 命令	1	1
$\overline{\text{INT}}$ 割り込み処理	0	0
$\overline{\text{NMI}}$ 割り込み処理	0	変化なし
RETN 命令	IFF_2 の状態がコピーされる	変化なし
RETI 命令	変化なし	変化なし

0：$\overline{\text{INT}}$ 割り込み受け付け禁止，
1：$\overline{\text{INT}}$ 割り込み受け付け許可

であれば基本的にはどんな命令でもよいのですが，プログラムの分岐およびプログラム・カウンタのスタックへの待避を考えると，RST命令かCALL命令が適します．

RST命令は8種類あり，それぞれアドレス0000h，0008h，0010h，0018h，0020h，0028h，0030h，0038hをコールしプログラムを実行します．RST命令は1バイト命令なので，データ・バス上に命令を出力する回路が簡単に実現できます．$\overline{\text{INT}}$ 割り込みの種類が8種類以下の場合便利です．

CALL命令は3バイト命令なので，回路が複雑になるのでここでは取り上げません．

● **モード0ではRST命令が最適**

図11の例に示すように，RST命令を割り込みベクタとして使った場合は，CPUはこれを読み込み実行します．RST命令はM1サイクルを5クロックで実行し，また割り込みベクタ読み込み時は，CPUが強制的に2クロックのウェイトを入れるので，合計7クロックの時間がかかります．

この割り込みベクタを読み込むときの動作は，$\overline{\text{MREQ}}$ に替わって $\overline{\text{IORQ}}$ がアクティブになり，それ以外のタイミングは通常のオペコード・フェッチと同等であることから，スペシャルM1サイクルとも呼ばれています．

スペシャルM1サイクル後は，通常のRST命令の実行と同様，プログラム・カウンタをスタックに退避するためにメモリ・ライト・サイクルが2回実行されます．

そしてRST命令の飛び先である各アドレスからオペコード・フェッチを開始し，割り込み処理ルーチンの実行に移ります．

● **モード0割り込み制御回路例**

モード0の $\overline{\text{INT}}$ 割り込みを発生させる外部回路を図12に示します．CPUのピン番号は40ピンDIPタイプの番号を使用しています．

HCタイプの汎用ロジックICを数個使用して，8種類の割り込み入力の優先順位付けと，RST命令の発生を行っています．

RST命令は8種類あるので，割り込みも8種類に対応できるのですが，RST 00Hは，リセット後と同様のアドレス0000hからの実行なので通常はあまり使用しません．ここでもRST 00Hは使用していません．

動作確認のためのLED点灯回路は図10の回路をそのまま使います．

スイッチ回路出力(HC74の8番ピン)をRST 08H～RST 38Hのどれかに接続してスイッチを押すと，対応した割り込みが発生します．

プライオリティ・エンコーダの入力に接続されている割り込み保持用のフリップフロップは，エンコードされた3ビットのコードと，CPUが出力する割り込みベクタ要求信号($\overline{\mathrm{M1}}$と$\overline{\mathrm{IORQ}}$がLレベルになったとき)信号でリセットしています．

● **モード0割り込み処理プログラム**

実際のプログラムでは**リスト2**のように，割り込み処理が2ステップ程度ですむわけがないので，各RST命令のジャンプ先アドレスには割り込み処理ルーチンへのジャンプ命令を書くのが普通です．

プログラムの動作を説明します．まずリセット直後にスタックを設定し，LED点滅データ保持用のAレジスタの値をクリアします．そして割り込みのモードをモード0に設定し割り込み受け付けを許可します．そして無限ループさせているだけです．

この状態で，外部からの$\overline{\mathrm{INT}}$割り込みが発生すると，プログラム実行アドレスはハードウェア的に分岐され，それぞれのRST命令の番地から実行が開始されます．

それぞれの割り込み処理ルーチンでは，反転表示したいビットに相当するBレジスタのビットを1にして，LED点灯出力ルーチンへジャンプします．

〈リスト2〉 $\overline{\mathrm{INT}}$割り込み(モード0)で七つのLEDを反転表示するプログラム

```
        org     0000h
START:  jp      JMP     ;スタート

RST08H:
        org     0010h   ;RST 08H
        ld      b,02h
        jp      INTRET
RST10H:
        org     0010h   ;RST 10H
        ld      b,02h
        jp      INTRET
RST18H:
        org     0018h   ;RST 18H
        ld      b,04h
        jp      INTRET
RST20H:
        org     0020h   ;RST 20H
        ld      b,08h
        jp      INTRET
RST28H:
        org     0028h   ;RST 28H
        ld      b,10h
        jp      INTRET
RST30H:
        org     0030h   ;RST 30H
        ld      b,20h
        jp      INTRET
RST38H:
        org     0038h   ;RST 38H
        ld      b,40h
        jp      INTRET
INTRET:
        xor     B
        out     (20h),A ;LED点灯
        ei              ;割り込み受け付け許可
        reti            ;メイン・ルーチンへ戻る

        org     080H
JMP:    ld      sp,0    ;スタック設定
        ld      a,0
        im      0       ;割り込みモード0
        ei              ;割り込み受け付け許可
LABEL1:
        jp      LABEL1  ;無限ループ
```

そしてAレジスタとBレジスタの排他的論理和(XOR)を計算してからその結果を，LED点灯回路に出力します．$\overline{\mathrm{INT}}$割り込み処理の終了では，次の割り込みの受け付けを許可するために，EI命令を実行してからRETI命令でもとのアドレスに復帰します．

入力される割り込みベクタによって，点滅するビットが違うことを実験してみてください．

● **DI命令とEI命令と動作**

割り込み受け付けを禁止するのがDI命令です．では，DI命令を実行中に割り込みが発生したらどうなるでしょう．DI命令は，命令実行中は割り込みが発生してもこの受け付けを禁止します．つまりDI命令と次の命令以後の間で，割り込みルーチンにジャンプすることはありません．

またEI命令は割り込み受け付けを許可する命令です．しかし実はEI命令を実行してもすぐには割り込み受け付け許可状態にはなりません．これは，割り込みルーチンの最後に必要な，RETI命令の実行を考え，EI命令の次の命令の実行後から受け付け許可状態になります．

ここで連続的に割り込みが発生したときを考えてみます．まず割り込みが発生するとメイン・ルーチンへ

Q：Z80の割り込み制御で，とくにモード2は非常に特徴的なシステム構成だと思う．このような割り込みシステムに設計した理由は？ 8080からの拡張ならば，RST命令を増やすなどの方法でもよかったのではないか．

A：Z80CPUは，ファミリ周辺デバイスも含めたトータル・システムを目指して設計されました．それ故，ファミリ周辺デバイスからの割り込みを効率的に処理できるように，モード2割り込みが付け加えられたのだと思います．割り込みコントローラを使ったシステムの場合，割り込みルーチンのプログラムが作りにくいというのもあるでしょう．

また8080の1バイト命令の未使用命令は，もうほとんど残っていなかったので，さらに増やすとなると2バイト以上のRST命令になってしまいます．

以上から，外部から1バイトのベクタを読み込み，間接的にジャンプするこの方式が，周辺デバイスに付加する回路規模からも現実的だったのだと思います．

の戻りアドレスをスタックにPUSHし，割り込みルーチンに実行が移ります．そしてルーチンが終わったらスタックから，次の割り込み処理ができるようにEI命令を実行し，元のメイン・ルーチンの続きを実行するためRETI命令を実行します．

このとき，EI命令を実行した瞬間から割り込み受け付けが許可状態になると，次のRETI命令を実行しないうちに再度割り込みルーチンの先頭にジャンプしてしまいます．これをそのまま繰り返すと，RAMがスタックでいっぱいになってしまいます．

では割り込みルーチンからメイン・ルーチンに戻るRETI命令に，EI命令と同等な動作を入れれば問題ないように思えます．

しかしRETI命令に，割り込み受け付けを許可する動作を加えると，後述する多重割り込みの処理で不都合が出る場合があります．一般的な割り込み処理ルーチンの最後では，面倒でもEI命令とRETI命令を入れるようにします．

またEI命令の次の命令の実行後から割り込み受け付けが許可されますから，EI命令とDI命令が続いた場合は，割り込み受け付けは禁止のままとなります．EIとDIの間にNOP命令が1個でもあれば，そこで割り込みを受け付けることができます．

このEI命令とDI命令の動作は，モード1でも2でも同じです．

モード1割り込みの動作

● モード1割り込みとは

モード1割り込みの考え方は$\overline{\text{NMI}}$と同じで，割り込み信号が入力されたら特定のアドレスへジャンプするモードです．もしZ80を使用して組み上げるシステムの割り込み処理が一つでよい場合は，モード1を使用するのも得策です．

また2種類使用する場合でも，片方は$\overline{\text{NMI}}$割り込みを使用するのがわかりやすくてよいでしょう．

● モード1割り込みのバスの動き

モード1の$\overline{\text{INT}}$受け付け後の応答サイクルは，タイミング的には図11のモード0の例とまったく同じです．ただし，読み取った割り込みベクタつまりオペコードは，どんなデータであろうとすべて無視し，RST 38H(オペコード：FFh)という命令に置き換えて実行されます．つまりモード1はマスク可能な$\overline{\text{NMI}}$のアドレス0038h版とも考えられます．

Z80七不思議　割り込み編

Q：周辺LSIがRETI命令を認識する必要はないのではないか．またこのことによって，CPUと周辺LSIの間にバッファを入れる場合など，回路が面倒になってしまう．

A：Z80ファミリは，周辺も含めて一つのシステムを構成するという考えに立って設計されたのだと思います．また，割り込み優先順位に関しても，その優先順位をダイナミックに変えなければならないアプリケーションはほとんどないと考えて，割り込み優先順位はデイジィ・チェーンで固定し，Z80周辺LSIはRETI命令を受けて周辺デバイス間で優先順位を制御するという方式をとっています．

そのため，複数のボードにまたがった場合のハードウェアが面倒になるというデメリットより，周辺デバイスに割り込みの終了を認識させることにより処理の簡素化を図り，ソフトウェアのオーバ・ヘッドを減らすほうを取ったのだと思います．

Q：モード2の割り込みベクタが偶数アドレスでなければならないのはなぜか．他のCPUでは割り込み本数を稼ぐため，シフトしてから使用するものなどがある．

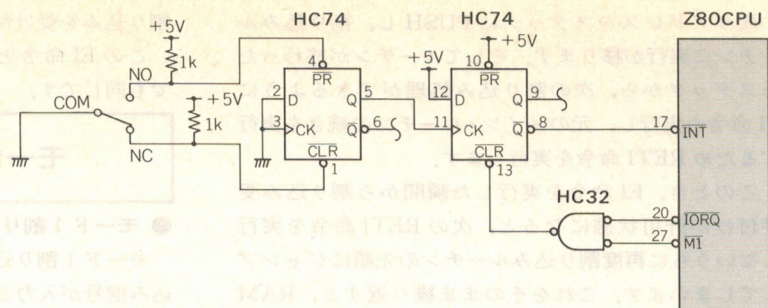
A：これも，できるだけゲート数を減らすためではないかと思います．

またZ80設計当時，128本以上の割り込みベクタが必要になるアプリケーションはほとんど存在しなかったのではないかと思います．ベクタが格納されているアドレスをシフト計算なしですぐに知ることができる，というメリットもあります．

Q：割り込みベクタ要求時に，なぜ$\overline{\text{IORQ}}$がアクティブになるのか．割り込みベクタ・リクエストなどのような専用信号線を入れたほうがわかりやすいし，回路が簡単になったのではないか．

A：これも信号線の節約というのが理由と思います．Z80PIOのリセット入力やZ80SIOのボンディング・オプションなどをみるとわかるように，当時はパッケージの自由度が少なく，40ピンなら必ず40ピンに収めなければならなかったためだと思われます．

〈信垣育司〉

〈図13〉
$\overline{\text{INT}}$割り込み(モード1)を発生させる回路

そこでモード1で使用する場合は，基本的にはデータ・バスに割り込みベクタを出力しなくても，$\overline{\text{INT}}$入力にLレベルの信号を入力するだけでアドレス0038hへジャンプさせることができます．

バスの動作としてはモード0と同じタイミングで動いています．スペシャルM1サイクルを実行後，プログラム・カウンタの上位，下位をスタックに待避し，アドレス0038hにジャンプします．

割り込み応答サイクルに挿入されている2クロックのウェイトや，$\overline{\text{IORQ}}$と$\overline{\text{M1}}$による割り込みベクタ要求信号は，本来モード1割り込みではバスからは何の情報も読み込まないので必要ありません．しかしモード0，モード2とバス・サイクルを統一している(ハードウェアが共通になり，回路規模を小さくすることができる)ためだと思われます．

● モード1割り込み制御回路例

モード1の$\overline{\text{INT}}$割り込みを発生させる簡単な外部回路を図13に示します．CPUのピン番号は40ピンDIPタイプの番号を使用しています．

基本的な考え方は図9の回路とあまり変わりありませんが，手動のスイッチの接点出力をSR-FF(セット/リセット・フリップフロップ)でチャタリング除去処理したものを，割り込み保持用のフリップフロップの入力としています．

この割り込み保持用のフリップフロップは，スイッチの押し下げでセットされ，CPUの割り込みベクタ要求信号($\overline{\text{M1}}$と$\overline{\text{IORQ}}$が同時にLレベルになる)でリセットしています．

また動作確認のためのLED点灯回路は，これも図10の回路を使います．

● モード1割り込みの処理プログラム

いままでの例と同様に，モード1割り込みでLEDを反転表示するプログラムを，リスト3に示します．

プログラムの動作を説明します．まずリセット直後にスタックの設定と表示データを保持するAレジスタをクリアします．そして割り込みモードをモード1に設定し割り込み受け付けを許可します．

そして無限ループを繰り返すだけで，他には何もしていません．

外部からの$\overline{\text{INT}}$割り込みが発生すると，プログラム実行アドレスはハードウェア的に分岐され，アドレス0038hから実行を始めます．

Aレジスタと02hとの排他的論理和を計算し，ビット1を反転させたあとにLED点灯回路に出力します．またモード0と同様，次の割り込み受け付け許可のためにEI命令を実行し，RETI命令で元のプログラムに戻ります．

● モード1での多重割り込み

モード0では割り込みベクタとしてRST命令を使うことで，8本の割り込みを制御することが可能でした．多重割り込みについても，それぞれの割り込みルーチンの先頭で，優先順位のフラグを制御すれば可能です．

ではアドレスが0038hに固定されているモード1の場合はどうすればよいでしょうか．これも基本的な考え方は同じです．図14に割り込みモード1における多重割り込み制御の流れを示します．

モード1ですから，タイマ割り込みもプリンタの割り込みも，すべて同じアドレスにジャンプしてきます．これではプログラムでは何の割り込みが発生したかわからないので，各割り込み源のステータスをチェックし，何の割り込みが発生したかを調べます．ステータスを調べる順序は優先順位の高い順から調べます．

モード2割り込み

● モード2割り込みとは

モード2割り込みはZ80の三つの割り込みモードの中でもっとも強力なモードです．

割り込み処理ルーチンは最大128種類までハードウェア的に分岐することが可能です．

分岐の方法としては，割り込みを発生したコントローラが，CPUが割り込みベクタ要求を発生するタイミングで，1バイトの割り込みベクタをデータ・バスに出力することで実現します．この動作そのものはモード0やモード1と同じです．

図15にモード2割り込みの割り込み処理ルーチンへの分岐手順を示します．

〈リスト 3〉 $\overline{\text{INT}}$ 割り込み(モード 1)で LED を反転表示するプログラム

```
        org     0000h
START:                  ;スタート
        ld      sp,0    ;スタック設定
        ld      a,0
        im      1       ;割り込みモード1
        ei              ;割り込み受け付け許可
LABEL1:
        jp      LABEL1  ;無限ループ


        org     0038h
INTTOP:                 ;RST 38H モード1割り込み
        xor     02h
        out     (20H),a ;LED点灯
        ei              ;割り込み受け付け許可
        reti            ;メイン・ルーチンへ戻る
```

モード 0 割り込みの RST 命令と比べると，分岐できるアドレスが自由になり，しかも 128 種類の割り込みに対応できるなど，かなり強化されています．

また割り込みベクタ・テーブルを RAM 領域に設定することも可能で，割り込み処理ルーチンのアドレスの書き換えをプログラムで行えば，128 種類以上の分岐も可能となります．

● デイジィ・チェーンによる割り込み優先順位

モード 2 割り込みのさらなる特徴として，割り込み優先順位をハードウェアで決定できることがあげられます．

Z80 周辺 LSI では，デイジィ・チェーンという手法でこの優先順位の管理を実現しています．これは図 16 に示すように割り込みを発生させる LSI の制御端子を，割り込みの優先順に直列に接続し，自分より下位の LSI に対し割り込み禁止を伝達します．

Z80 周辺 LSI には IEI(Interrupt Enable In)と IEO(Interrup Enable Out)という二つの端子が用意されており，これらを接続します．

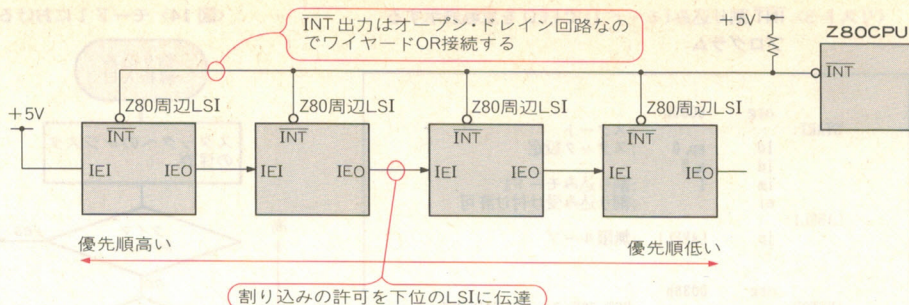
最も優先順位の高い LSI の IEI は＋5 V に接続します．各 LSI の伝搬遅延時間の関係で，あまり多くの LSI を接続できません．規格上は四つまでとなっています．外付け回路を設けることで，より多くの LSI を接続することも可能です．

〈図 14〉 モード 1 における多重割り込みの流れ

● Z80 周辺 LSI とモード 2 割り込み

さらに Z80 周辺 LSI は，割り込み処理終了の情報を，周辺 LSI がデータ・バスを常時監視し判断することで，自動的に割り込み要求をとりさげるように設計されています．この割り込み処理終了を周辺 LSI に

〈図 15〉
割り込み処理ルーチン実行の手順

〈図16〉
Z80周辺LSIのテイジィ・チェーン接続

〈リスト4〉割り込みモード，ベクタ設定のプログラム例

```
        DI
        LD      A,HIGH PIO_INT_A
        LD      I,A                 ;割り込みベクタ設定
        IM      2                   ;割り込みモード2
        LD      SP,0                ;スタックの設定

        ORG     1000H               ;割り込みベクタ・テーブル
PIO_INT_A:
        DEFW    PIO_A               ;
PIO_INT_B:
        DEFW    PIO_A               ;
```

(a)ORGで直接アドレスを指定

```
        ORG     ($+1) AND 0FFFEH ;割り込みベクタ・テーブル
PIO_INT_A:
        DEFW    PIO_A               ;
PIO_INT_B:
        DEFW    PIO_A               ;
```

(b) 算術演算子などで偶数アドレスに自動設定

〈リスト5〉割り込みベクタ・テーブルのアドレスに注意

```
        DI
        LD      A,HIGH PIO_INT_A
        LD      I,A                 ;割り込みベクタ設定
        IM      2                   ;割り込みモード2
        LD      SP,0                ;スタックの設定

        ORG     ($+1) AND 0FFFEH ←(この結果が1FFEhのとき)
PIO_INT_A:
        DEFW    PIO_A  ←(1FFEhに格納される)
PIO_INT_B:
        DEFW    PIO_A  ←(2000hに格納される)

        END
```

明示するために，RETI命令があります．

RETI命令は，周辺LSIに対して割り込み処理ルーチンの終了を知らせる役目があり，Z80周辺LSIでモード2を使う場合には非常に重要な命令ですが，プログラム的には単なるサブルーチンから復帰するRET命令と同じです．この点はRETN命令とは逆です．

このRETI命令の監視により8259など割り込みコントローラを使った場合の，割り込み終了を周辺LSIに設定するプログラムが不要になります．

このようにZ80周辺LSIは，Z80CPUのモード2割り込みを強力にサポートしているため，Z80CPUと組み合わせて使用することでその能力を十分に発揮します．逆にそのほかのCPUと組み合わせて使用することが難しくなってしまい，汎用性という点では他の周辺LSIに一歩譲る感じになっています．

● モード2割り込みのバスの動作

モード2割り込み発生時のバスの動作を図17に示します．$\overline{INT}$割り込み認識後，モード0やモード1と同様，7クロックのスペシャルM1サイクルを実行します．

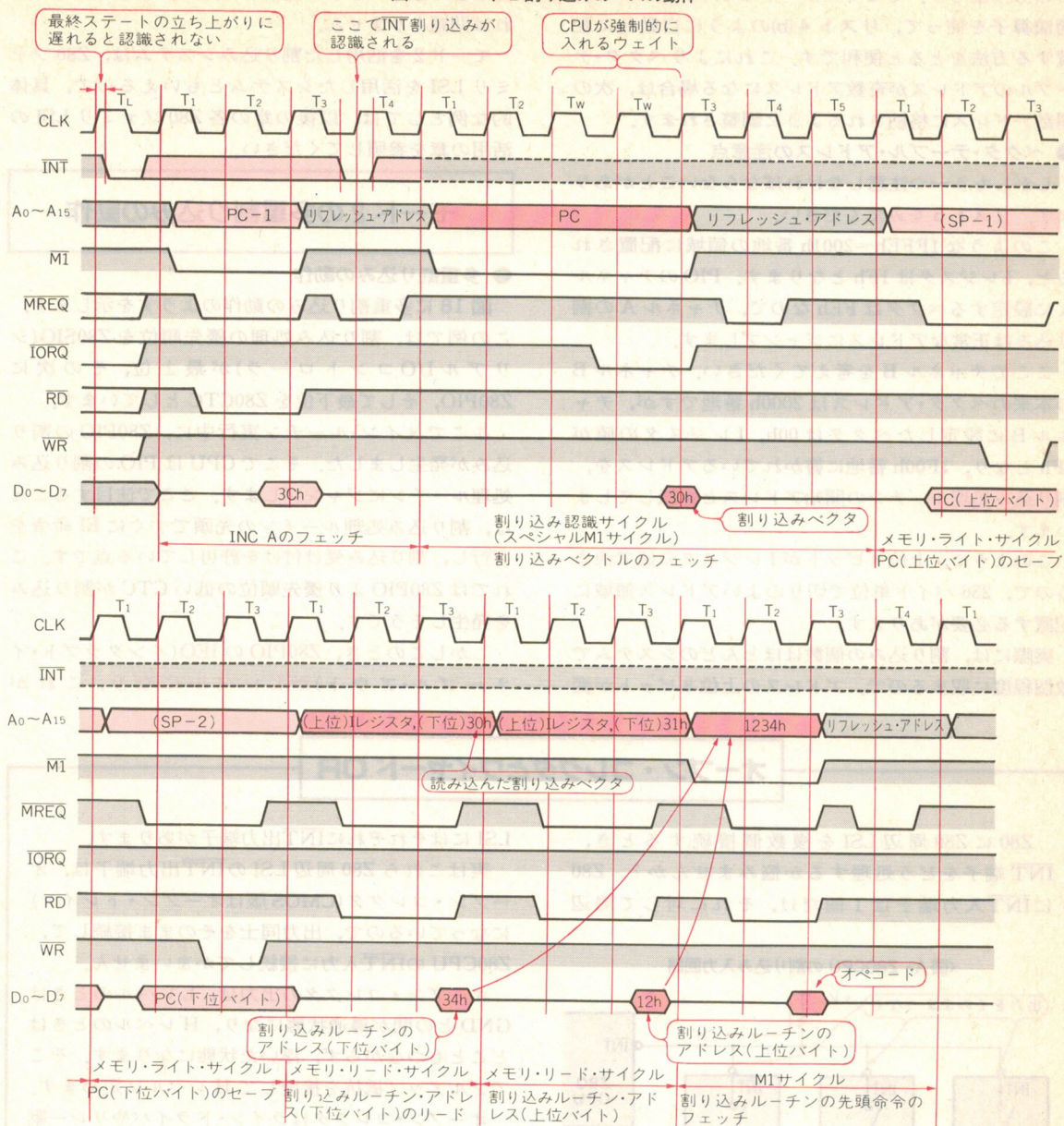
スペシャルM1サイクルでは$\overline{M1}$と$\overline{IORQ}$がLレベルになることで，CPUが割り込みベクタを読み込むわけですが，この割り込みベクタは，モード0のようなZ80が実行できる命令ではありません．いわゆるベクタ・アドレスになります．

CPUは図17のように割り込みを発生したデバイスから，割り込みベクタと呼ばれるデータを読み取ると，割り込み処理が終了して再び本来実行していたプログラムに復帰できるように，PCの値をスタックに積みます．

そして，先ほど読み取った割り込みベクタの値を下位8ビット，そしてI(インタラプト・ベクタ)レジスタの値を上位8ビットとした16ビットのアドレス，割り込みベクタ・テーブル・アドレスを生成します．

そしてこの割り込みベクタ・テーブル・アドレスのメモリの値を下位8ビット，割り込みベクタ・テーブル・アドレス+1のアドレスのメモリの値を上位8ビットとした16ビットのアドレスを，割り込み処理ルーチンのアドレスとして読み込みます．

こうして割り込み処理ルーチンの先頭にジャンプし，

〈図17〉モード2割り込みのバスの動作

割り込みプログラムが動き出します．

割り込み処理プログラムが終了したら，次の割り込みに備えて割り込み受け付けを許可するためにEI命令を実行し，割り込み発生前のアドレスに復帰するためにRETI命令で戻ります．これがモード2割り込みの大まかな流れです．〈末木　豊〉

● **割り込みモード2の割り込みプログラム**

まず割り込みベクタ・テーブルをどこに配置するかという問題があります．16ビットのアドレスのうち，上位8ビットはIレジスタで，下位8ビットは割り込みを発生したデバイスが出力するので，基本的には64Kバイトの空間の好きな部分に配置することができます．

リスト4(a)に一般的なモード2割り込み処理のベクタ設定と初期化ルーチン例を示します．この例のように，ORG命令で任意のアドレスに指定する方法もあります．しかしこの例では，プログラムの容量が増加したとき，割り込みベクタ・テーブルにプログラム領域が重なってしまうかもしれません．

当然，そのときは割り込みベクタ・テーブルのORGの値をずらせばすみますが，いちいちプログラムがどのアドレスまで使うかを気にしなければならないとい

うのも不便です．そこでロケーション・カウンタと算術演算子を使って，**リスト4**(b)のように自動的に配置する方法をとると便利です．これによりベクタ・テーブルのアドレスが奇数アドレスになる場合は，次の偶数アドレスに格納されるように調整されます．

● **ベクタ・テーブル・アドレスの注意点**

しかしもう一つ注意しなければならないことがあります．**リスト5**をみてください．

このような1FFEh～2001h番地の領域に配置されると，Iレジスタは1Fhとなります．PIOのチャネルAに設定するベクタはFEhなので，チャネルAの割り込みは正常なアドレスにジャンプします．

ここでチャネルBを考えてください．チャネルBの本来のベクタ・アドレスは2000h番地ですが，チャネルBに設定したベクタは00h，Iレジスタの値が1Fhとなり，1F00h番地に書かれているアドレスを，割り込み処理ルーチンの開始アドレスと認識してしまいます．

このように，上位8ビットがIレジスタの値で決まるので，256バイト単位で切りのよいアドレス領域に配置する必要があります．

実際には，割り込みの個数はほとんどのシステムで数個程度に収まるので，アドレスの上位8ビットが変化しない数十バイトの連続したメモリ領域が確保できれば問題ありません．

モード2を活用した割り込みシステムは，Z80ファミリLSIを活用したシステムともいえるので，具体的な例としては，以後の章の各Z80ファミリLSIの活用の章を参照してください．

モード2の多重割り込みの動作

● **多重割り込みの動作**

図18に多重割り込みの動作のようすを示します．この例では，割り込み処理の優先順位をZ80SIO(シリアルI/Oコントローラ)が最上位，その次にZ80PIO，そして最下位をZ80CTCとしています．

ここでメイン・ルーチン実行中に，Z80PIOの割り込みが発生しました．そこでCPUはPIOの割り込み処理ルーチンにジャンプします．ここで注目することは，割り込み処理ルーチンの先頭ですぐにEI命令を実行し，割り込み受け付けを許可している点です．これではZ80PIOより優先順位の低いCTCが割り込みを発生しそうです．

しかしこのとき，Z80PIOのIEO(インタラプト・イネーブル・アウト)がLレベルになり，これが

オープン・コレクタとワイヤードOR

Z80にZ80周辺LSIを複数個接続するとき，$\overline{\mathrm{INT}}$端子をどう処理するか悩みませんか？ Z80に$\overline{\mathrm{INT}}$入力端子は1個だけ，それに対して周辺LSIにはそれぞれに$\overline{\mathrm{INT}}$出力端子があります．

〈図A〉 Z80CPUの割り込み入力回路

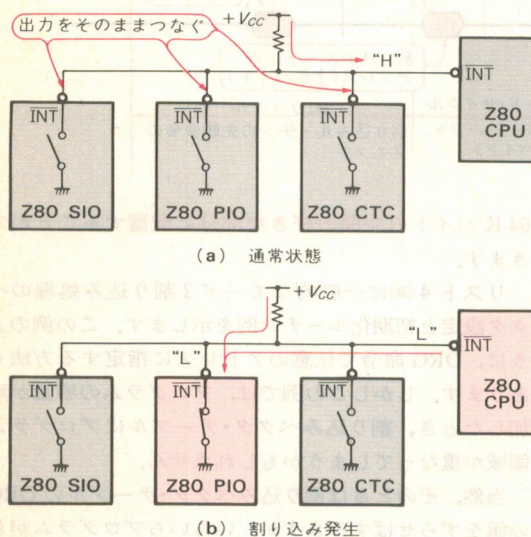
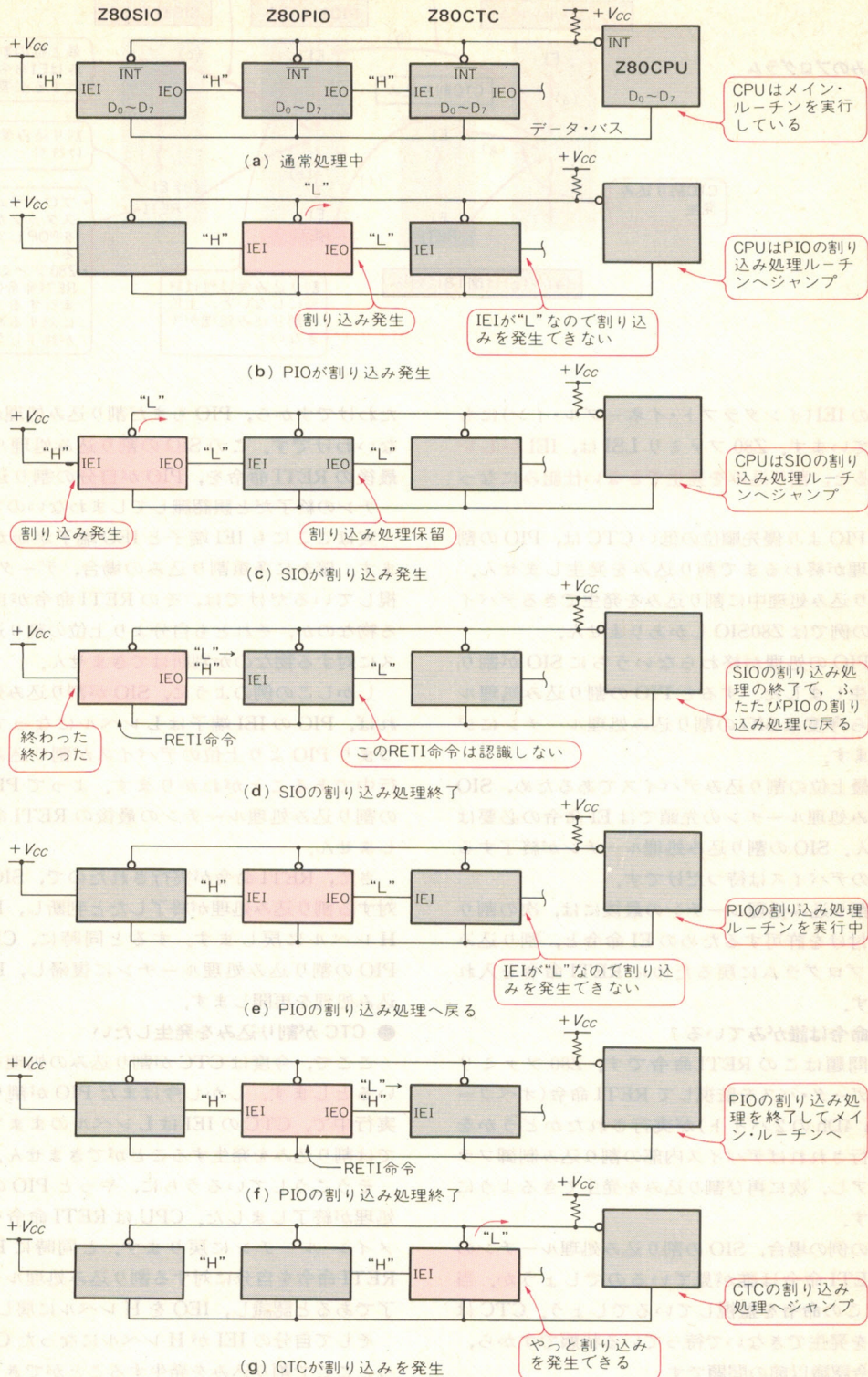
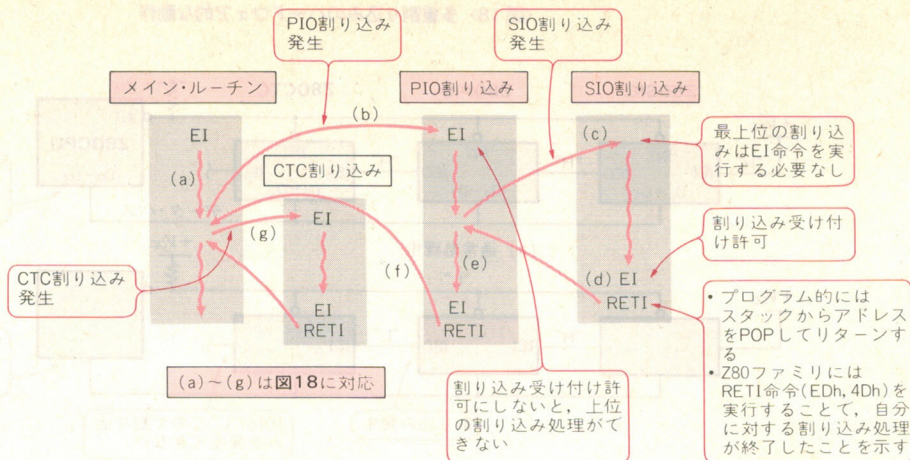
(a) 通常状態

(b) 割り込み発生

実はこれらZ80周辺LSIの$\overline{\mathrm{INT}}$出力端子は，オープン・コレクタ(CMOS版はオープン・ドレイン)になっているので，出力同士をそのまま接続して，Z80CPUの$\overline{\mathrm{INT}}$入力に接続してかまいません．

オープン・コレクタの出力は，LレベルのときはGNDとの間が導通状態になり，Hレベルのときはどこともつながらず，浮いた状態になります．そこでプルアップ抵抗を接続してHレベルに定めます．

オープン・コレクタはライン・ドライバやリレー駆動など，インターフェース回路によく用いられます．ダーリントン・トランジスタ・アレイもその仲間です．汎用ロジックICとしては，LS05，LS06，LS07などがオープン・コレクタ出力です．

オープン・コレクタは自分自身からHレベルを出しませんから，出力同士をそのまま接続できます．その場合，どれかの出力がLレベルになれば共通出力がLレベルになるわけで，負論理のOR回路と同じ動作になります．これを特にワイヤードOR回路と呼び，リセット回路や割り込み回路など不特定個数の信号源を想定した回路に用いられます(図A)．

〈図18〉多重割り込みのハードウェア的な動作

〈図19〉
多重割り込みのプログラム的な動作

Z80CTCのIEI(インタラプト・イネーブル・イン)にも接続されています．Z80ファミリLSIは，IEIがLレベルになると，割り込みを発生できない仕組みになっています．

よってPIOより優先順位の低いCTCは，PIOの割り込み処理が終わるまで割り込みを発生しません．PIOの割り込み処理中に割り込みを発生できるデバイスは，この例ではZ80SIOしかありません．

さて，PIOの処理が終わらないうちにSIOが割り込みを発生しました．するとPIOの割り込み処理ルーチンから今度はSIOの割り込み処理ルーチンにジャンプします．

SIOが最上位の割り込みデバイスであるため，SIOの割り込み処理ルーチンの先頭ではEI命令の必要はありません．SIOの割り込み処理ルーチンが終了するまでほかのデバイスは待つだけです．

SIOの割り込み処理ルーチンの最後には，次の割り込み受け付けを許可するためのEI命令と，割り込み発生前のプログラムに戻るためにRETI命令を入れておきます．

● RETI命令は誰がみている？

さて，問題はこのRETI命令です．Z80ファミリLSIは，データ・バスを監視してRETI命令(オペコードはEDh 4Dhの2バイト)が実行されたかどうかを調べ，実行されればデバイス内部の割り込み制御フラグをクリアし，次に再び割り込みを発生できるように準備します．

ではこの例の場合，SIOの割り込み処理ルーチンの最後のRETI命令は誰が見ているのでしょうか．当然SIOはこの命令を監視しているでしょう．CTCは割り込みを発生できないで待っている状態ですから，RETI命令認識以前の問題です．

問題はPIOです．このSIOの割り込み処理が発生する前は，PIOの割り込み処理ルーチンを実行していたわけですから，PIOもまだ割り込み処理が終わってないわけです．このSIOの割り込み処理ルーチンの最後のRETI命令を，PIOが自分の割り込み処理ルーチンの終了だと誤認識してしまわないのでしょうか．

実はここにもIEI端子とIEO端子がかかわってきます．確かに多重割り込みの場合，データ・バスを監視しているだけでは，そのRETI命令が自分に対する物なのか，それとも自分より上位の割り込みデバイスに対する物なのか判断はできません．

しかしこの例のように，SIOが割り込み処理中であれば，PIOのIEI端子はLレベルになっています．つまりPIOより上位のデバイスが割り込み処理を実行中であることがわかります．よってPIOは，SIOの割り込み処理ルーチンの最後のRETI命令は認識しません．

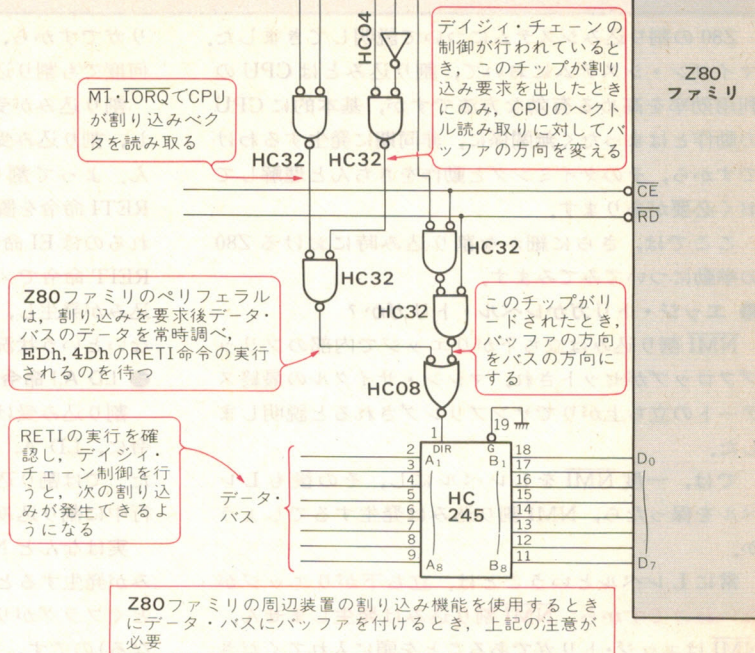
さて，RETI命令が実行されたので，SIOは自分に対する割り込み処理が終了したと判断し，IEO端子をHレベルに戻します．すると同時に，CPUは再びPIOの割り込み処理ルーチンに復帰し，PIOの割り込み処理を再開します．

● CTCが割り込みを発生したい

ここで，今度はCTCが割り込みの処理を要求しているとします．しかし今はまだPIOが割り込み処理実行中で，CTCのIEIはLレベルのままです．これでは割り込みを発生することができません．

そうこうしているうちに，やっとPIOの割り込み処理が終了しました．CPUはRETI命令を実行してメイン・ルーチンに戻ります．と同時にPIOはそのRETI命令を自分に対する割り込み処理ルーチンの終了であると認識し，IEOをHレベルに戻します．

そして自分のIEIがHレベルになったCTCは，やっとここで割り込みを発生することができるのです．図18の動作をプログラム的にみたのが図19です．よく見比べてみてください．

〈図20〉
割り込み機能使用時のバッファ・コントロール

● EI命令とRETI命令

ここで，各割り込みルーチンの最後のEI命令やRETI命令の動作について確認します．

EI命令は割り込みを許可する命令です．これを実行しないと，いくらデバイスが割り込みを発生して$\overline{INT}$端子をLレベルにしても，CPUはこれを無視して処理を続けます．

RETI命令は，CPUについてだけみると，通常のRET命令の動作となんら変わりません．スタックに積まれたアドレスをPOPして，そのアドレスに戻るだけです．

しかしRETI命令は，Z80ファミリLSIにとっては，自分が発生した割り込みのルーチンの終了を判断する大切なキーワードとなります．

もしコーディング時のミスで，RETI命令を単にRETと記述してしまっても，CPU単体としての動作はRETIもRETも同じですから，割り込み処理ルーチンを終了して元のアドレスに復帰します．

しかし，Z80ファミリはRETI命令が実行されていないので，自分の割り込み処理ルーチンが終了したと判断できず，CPUがメイン・ルーチンに復帰しても，自分の割り込み処理ルーチンを続けていると認識し，次の割り込みを発生しません．

またデイジィ・チェーン接続で多重割り込みができる場合は，そのデバイスより優先順位の低いデバイスも，割り込みを発生できなくなってしまいます．

● CPUとZ80ファミリLSI間にバッファを入れた場合

最後に，Z80CPUに多数の周辺LSIを接続したときの注意点です．Z80ファミリにかぎらず，ロジックICは一つの出力からいくつものICの入力へは接続できません．そのため，多数のI/Oを増設する場合など，バスの駆動能力が低下し誤動作の原因になります．

そこでデータ・バスやアドレス・バスに，HC245などのバッファICを接続し，多数の周辺ICを接続してもCPUがすべてのICに正しくデータが送られるようにします．そしてCPUがデータを読み込むときは外からCPUへデータが向かう方向へ，書き込むときはCPUから外へ向かう方向に制御します．

ここで注意点があります．これまで説明したようにZ80CPUとそのファミリは，割り込みベクタやRETI命令の認識など，単純なデータの入出力以外にも情報のやり取りをしています．このためHC245などのバッファを間に入れる場合は，図20のように$\overline{IORQ}$や$\overline{M1}$，$\overline{IEO}$や$\overline{IEI}$などの信号の状態をも考慮して，バッファの方向を制御してやる必要があります．

〈野口智樹〉

（トランジスタ技術1994年6月号および10月号に加筆，修正）

Appendix

Z80の割り込み動作の詳細

磯沼 薫

Z80の割り込みシステムについて説明してきました．マイコン・システムにおいて，割り込みとはCPUの利用効率を高める有効な方法ですが，基本的にCPUの動作とはまったく無関係に，非同期に発生するわけですから，そのタイミングと動作をきちんと理解しておく必要があります．

ここでは，さらに細かな割り込み時におけるZ80の挙動についてみてみます．

● エッジ・トリガかレベル・トリガか？

$\overline{\text{NMI}}$割り込みは立ち下がりエッジで内部のフリップフロップがセットされ，マシン・サイクルの最終ステートの立ち上がりでサンプリングされると説明しました．

では，一度$\overline{\text{NMI}}$をLレベルにし，その後もLレベルを保ったら，$\overline{\text{NMI}}$割り込みは発生するでしょうか．

常にLレベルということは，立ち下がりエッジがないわけですから，$\overline{\text{NMI}}$割り込みは発生しません．$\overline{\text{NMI}}$はエッジ・トリガであることを頭に入れてください．

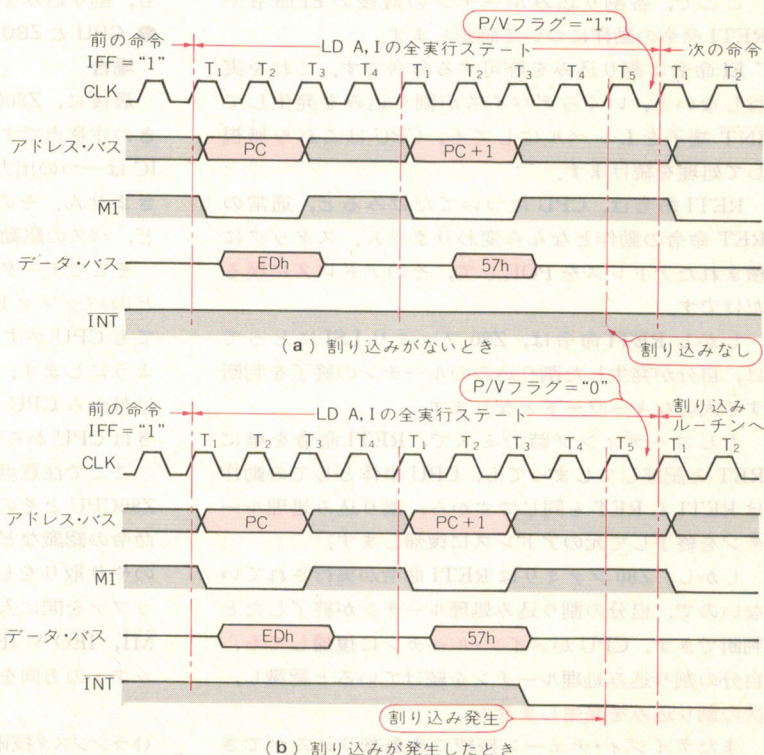
では，$\overline{\text{INT}}$はどうかというと，こちらはレベル・トリガですから，$\overline{\text{INT}}$端子を常にLレベルにすると，何度でも割り込みが発生します．

割り込みが受け付けられるためにはEI命令を実行し，割り込み受け付け許可状態にしなくてはなりません．よって割り込みルーチンの最後にはEI命令とRETI命令を置きます．割り込みが実際に受け付けられるのはEI命令の次の命令を実行してからですから，REIT命令でメイン・ルーチンに戻ったとたんに割り込みが発生し，メイン・ルーチンがちっとも実行されないという状況に陥ります．

● LD A,I命令実行時に割り込みが発生したら？

割り込み受け付け禁止状態か許可状態かを調べる命令に，LD A，I命令があることはすでに説明しました．では割り込み受け付け許可状態で，この命令を実行中に割り込みが発生したらどうなるでしょうか．

実はなんとNMOS版のZ80では，実行中に割り込みが発生すると，割り込み受け付け禁止中であるがごとくフラグがリセットしてしまう(P/Vフラグが0になる)のです．すべてについて確認したわけではありませんが筆者が実験した限りでは，どのメーカ製でもNMOS版ではこのようになるようです(図A)．

〈図A〉
LD A,I命令実行中に割り込みが発生すると

〈図B〉 Z80の$\overline{INT}$割り込み受け付けタイミング

(a) オペコード, オペランド読み込みのみのとき

(b) オペコード, オペランド読み込みの後にメモリ・アクセスが入る命令のとき

ではCMOS版ではどうかというと, ザイログのZ80や周辺LSIを集積したZ84C15/C11などではこのような誤動作はありません. 割り込み受け付け許可状態であれば, 必ずP/Vフラグがセットされます.

しかし同じCMOS版といえども, 東芝製のTMPZ84C015では誤動作しました.

また日立の64180についても, R1およびZバージョンからはマニュアルに「LD A,IおよびLD I,A命令実行中は, 割り込みをサンプリングしません.」と書かれています.

最後に最新のCPUである川崎製鉄のKL5C8012ではどうかというと, これも命令実行中に割り込みが発生しても誤動作はしないようです.

● 割り込みはどこでサンプリングされる？

Z80の割り込みは命令の最終ステートの立ち上がりでサンプリングされるとありますが, 本当でしょうか？

Z80の割り込み受け付けの瞬間のバスのようすを図Bに示します. 図B(a)のようにオペコードを読み込んで内部で実行するだけの命令のときはたしかにそうなのですが, 図B(b)のようにメモリの読み書きなどを伴う命令のときは, 最後ステートではなく, オペランドの読み込みの最終ステートの立ち上がりでサンプリングされるようです.

ちなみに64180ではこのような命令のときも, 割り

〈図C〉 64180の割り込み受け付けタイミング

込みのサンプリング・タイミングは最終ステートの2クロック前の立ち下がりとなります(図C).

● $\overline{NMI}$とDMA

$\overline{NMI}$は電源の異常やメモリのパリティ・エラーなど, エラーや緊急事態の割り込みに使うべきで, リセットに次ぐ最上位の割り込みだと説明しました.

しかしZ80では, $\overline{NMI}$とDMAとではDMAのほうが優先順位が高く, DMAがハングアップするとZ80CPUは手も足も出せなくなります.

64180では$\overline{NMI}$はDMAよりも優先順位が高く, $\overline{NMI}$割り込みでDMAを中止させるようなことも可能です.

もっとも最近ではZ80にDMAはほとんど使用しないので, あまり考える必要はないと思います.

第7章

スイッチ入力やLEDの点灯をコントロールする

Z80PIOの使い方

野口智樹

Z80PIOの概要

● Z80PIOとは

Z80PIO(Parallel Input Output)はZ80ファミリのLSIで8ビットのポートを2個もっている，パラレルI/Oコントローラです．CPU側から見たとき，2個のI/Oポート・チャネルA，Bと，その動作モードをコントロールするためのレジスタがチャネルA，Bそれぞれにあり，計4個のアドレスを占有します．

Z80PIOは次のような特徴をもっています．

・チャネルA/Bという二つの8ビット・ポート
・各チャネルとも以下の四つのモードから選択してモードをプログラムできる
　モード0：バイト出力モード
　モード1：バイト入力モード
　モード2：バイト入出力モード(チャネルAのみ)
　モード3：ビット・モード
・A，B各チャネルにハンドシェイク機能がある
・Z80CPUのモード2割り込みに対応
・チャネルB出力は，ダーリントン・トランジスタを駆動可能

● Z80PIOのブロック図，ピン配置

Z80PIOのブロック図を図1に，ピン配置を図2に，図3と表1に電気的特性と入出力端子のタイミングを示します．

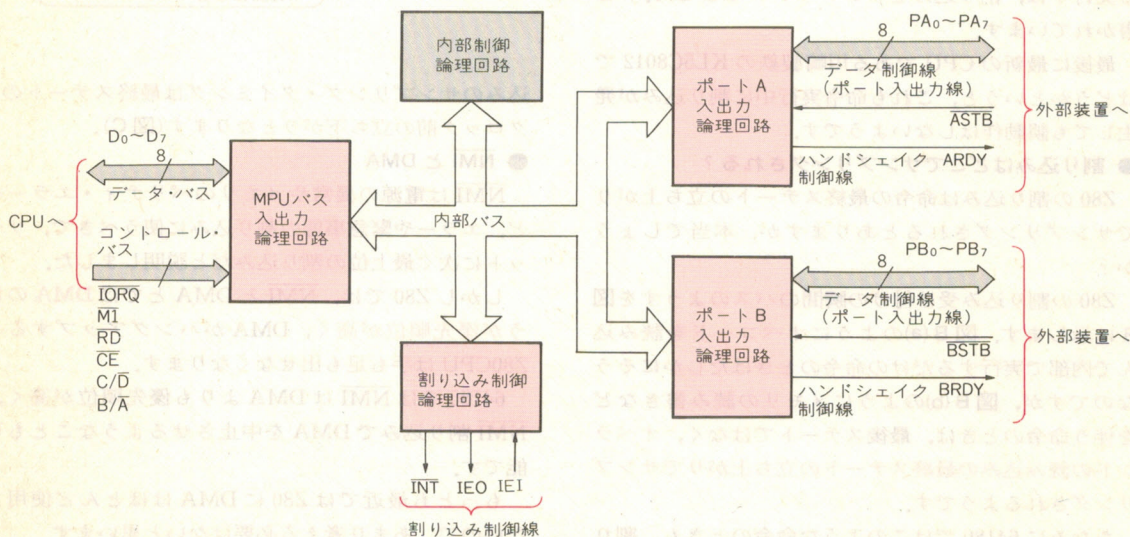
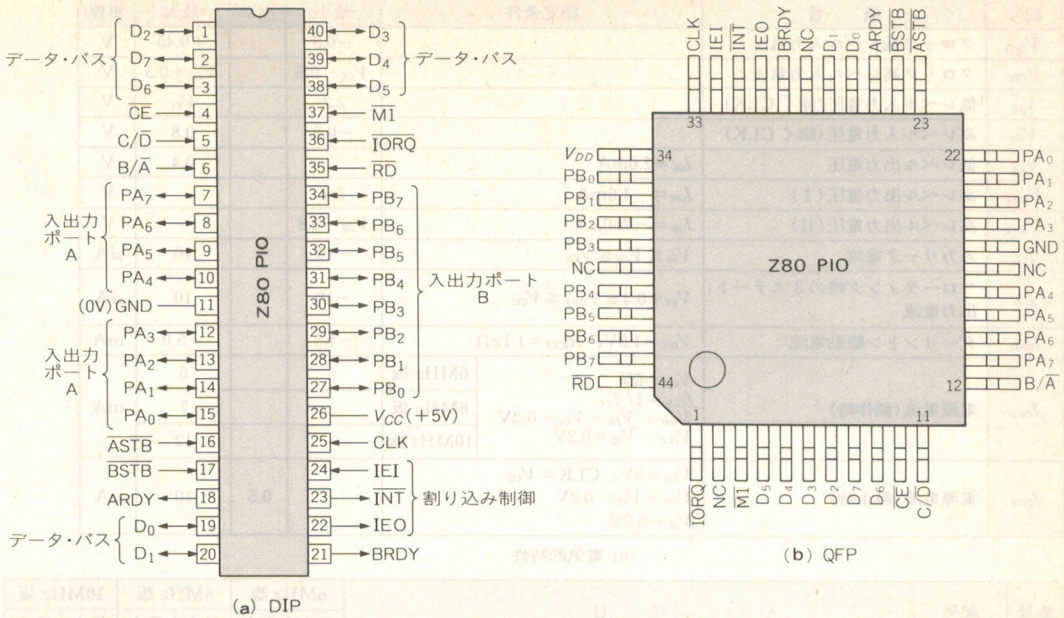
Z80CPUとの接続については，第4章で説明したのでそちらを参照してください．

実際に外部との接続に使われる端子は，PA_0～PA_7，$\overline{ASTB}$，ADRY，PB_0～PB_7，$\overline{BSTB}$，BRDY端子となります．

● Z80PIOのリセット

Z80PIOにはICパッケージのピン数の制限から，$\overline{RESET}$入力端子がありませんが，電源投入時に自動的に内部をリセットする，パワーONリセット機能が内蔵されています．また第4章のp.56ですでに説明したような外部回路を接続すれば，任意にリセットをかけることもできます．

またZ80PIOはリセットされるとポートA，Bとも，次の状態になります．

〈図1〉Z80PIOのブロック図

〈図 2〉 Z80PIO のピン配置

・割り込み禁止
・モード 1(入力)に設定
・データ入出力レジスタは全ビットがクリア
・マスク制御レジスタは全ビットがセット(マスク)
・ポート入出力線はハイ・インピーダンス状態
・RDY 端子は L レベル(ノット・レディ)

このリセット状態は，コントロール・ワードの設定があるまで保持されます．

Z80PIO の各種モード

まず Z80PIO にはどんなモードがあるのか，各モードの動作について説明します．

● Z80PIO のモード 0(バイト出力モード)

Z80PIO のモード 0 はバイト出力モードです．このモードに設定するとポートの方向は出力方向になります．図 4 にモード 0 におけるデータ出力のタイミングを示します．PIO の初期設定が終わり，データ・ポートに出力するデータを書き込むと，データ・ポートのデータが出力され，RDY 端子(実際には，チャネル A を使用するときは ARDY，チャネル B なら BRDY となる)が H レベルになります．RDY 端子は READY の意味で，準備完了を示す制御線です．

データを受け取る側では，この RDY 端子が H レベルになるのを待ち，H レベルになったらデータを読み込みます．そしてデータを受け取ったことを PIO に知らせるために，STB 端子(実際にはチャネル A を使用するときは ASTB，チャネル B なら BSTB となる)に読み出しクロックを入力します．

〈図 3〉 Z80PIO の入出力端子のタイミング

相手がデータを受け取らない，つまり STB 端子に読み出しクロックが入力されない限り，データ・ポートには書き込んだデータが出力されつづけ，RDY 端子は H レベルになったままで保持されます．

PIO は STB 端子の立ち上がりエッジを認識すると，RDY 端子を L レベル戻し，また INT を L レベルにして CPU に対して割り込みを発生します．

この割り込みは，相手がデータを受け取ったという

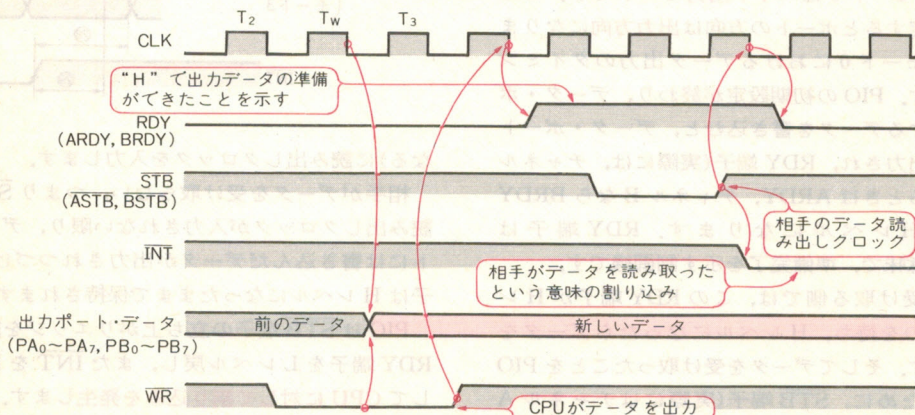
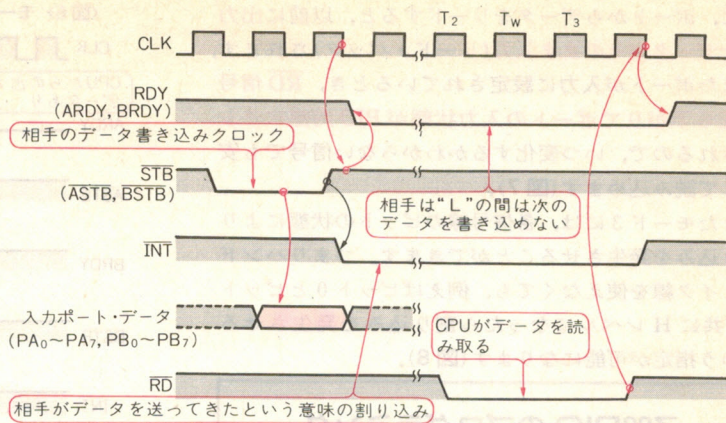
〈表 1〉 Z80PIO の電気的特性とタイミング

記号	項　目	測定条件	最小	標準	最大	単位
V_{ILC}	クロック低レベル入力電圧		−0.3		+0.45	V
V_{IHC}	クロック高レベル入力電圧		$V_{CC}-0.6$		$V_{CC}+0.3$	V
V_{IL}	低レベル入力電圧(除く CLK)		2.2		V_{CC}	V
V_{IH}	高レベル入力電圧(除く CLK)		−0.3		0.8	V
V_{OL}	低レベル出力電圧	$I_{OL}=2.0$mA			0.4	V
V_{OH1}	高レベル出力電圧(I)	$I_{OH}=-1.6$mA	2.4			V
V_{OH2}	高レベル出力電圧(II)	$I_{OH}=-250\mu$A	$V_{CC}-0.8$			V
I_{LI}	入力リーク電流	$V_{SS} \leqq V_{IN} \leqq V_{CC}$	−10		10	μA
I_{LO}	フローティング時の 3 ステート出力電流	$V_{SS}+0.4 \leqq V_{OUT} \leqq V_{CC}$	−10		10	μA
I_{OHD}	ダーリントン駆動電流	$V_{OH}=1.5$V, $R_{EXT}=1.1$kΩ	−1.5		−5.0	mA
I_{CC1}	電源電流(動作時)	$V_{CC}=5$V, $f_{CLK}=1/T_{CC}$, $V_{IHC}=V_{IH}=V_{CC}-0.2$V, $V_{ILC}=V_{IL}=0.2$V 6MHz 版			6	mA
		8MHz 版			7	mA
		10MHz 版			12	mA
I_{CC2}	電源電流(静止時)	$V_{CC}=5$V, CLK$=V_{CC}$, $V_{IH}=V_{CC}-0.2$V, $V_{IL}=0.2$V		0.5	10	μA

(a) 電気的特性

番号	記号	項　目	6MHz 版		8MHz 版		10MHz 版	
			最小	最大	最小	最大	最小	最大
19	$T_{CIO(C)}$	クロック立ち下がりに対する $\overline{IORQ}$=“H”のセットアップ時間(次のサイクルで REDY をアクティブにする場合)	170		140		120	
20	$T_{dC(RDYr)}$	クロック立ち下がりから RDY 立ち上がりまでの遅延		170		150		130
21	$T_{dC(RDYf)}$	クロック立ち下がりから RDY 立ち下がりまでの遅延		120		100		85
22	$T_{wSTB(C)}$	$\overline{STB}$ パルス幅	120		100		80	
23	$T_{sSTB(C)}$	クロック立ち下がりに対する $\overline{STB}$ の立ち上がりのセットアップ時間(次のサイクルで RDY をアクティブにする場合)	150		120		100	
24	$T_{dIO(PD)}$	$\overline{IORQ}$ 立ち上がりから出力データ確率までの遅延(モード 0)		160		140		120
25	$T_{sPD(STB)}$	$\overline{STB}$ 立ち上がりに対するセットアップ時間(モード 1)	190		140		75	
26	$T_{dSTB(PD)}$	$\overline{STB}$ 立ち下がりからの出力データまでの遅延(モード 2)		180		150		120
27	$T_{dSTB(PDr)}$	$\overline{STB}$ 立ち上がりからデータ・フロートまでの遅延(モード 2)		160		140		120
28	$T_{dPD(INT)}$	ポート・データ一致から $\overline{INT}$ 立ち下がりまでの遅延(モード 3)		430		360		200
29	$T_{dSTB(INT)}$	$\overline{STB}$ 立ち上がりから $\overline{INT}$ 立ち下がりまでの遅延		350		290		220

(b) Z80PIO の入出力端子のタイミング(単位：ns)

〈図 4〉 モード 0(バイト出力モード)のタイミング

〈図5〉
モード1(バイト入力モード)のタイミング

意味の割り込みと捉らえることができます．

● **Z80PIOのモード1**(バイト入力モード)

Z80PIOのモード1はバイト入力モードです．このモードに設定するとポートの方向は入力方向になります．図5にモード1におけるデータ入力のタイミングを示します．PIOの初期設定が終わり，データを受け取れる準備ができると，RDY端子はHレベル(受け取り準備完了)を示します．

RDY端子がHレベルであることを確認した相手は，データ・ポートにデータを出力し，データの準備できたことをPIOに示すために，$\overline{\mathrm{STB}}$端子に書き込みクロックを入力します．

PIOは，$\overline{\mathrm{STB}}$端子の立ち上がりクロックを認識すると，データ・ポートにデータが準備できたとみなし，RDY端子をLレベルにし，また$\overline{\mathrm{INT}}$をLレベルにしてCPUに割り込みを発生させます．

この割り込みは，相手がデータを送ってきたという意味の割り込みと捉らえることができます．

CPUは割り込み処理ルーチンで，PIOのポートからデータを読み込みます．CPUがデータを読み込むと，PIOはRDY端子をHレベルに戻し，次のデータ入力に備えます．

● **Z80PIOのモード2**(バイト入出力モード)

Z80PIOのモード2は，モード0とモード1を足したような動作をします．このモードに設定するとポートはハイ・インピーダンスになります．このモードはチャネルA専用で，次のような流れで動作するモードです．

▶ PIO出力

(1) CPUからPIOのチャネルAにデータを書き込む．
(2) そのデータがPIO内部にラッチされ，レディ(ARDY)がHレベルになる．
(3) 外部からのストローブ($\overline{\mathrm{ASTB}}$)信号がLレベルになると，PIO内部にラッチされていたデータが，チャネルAのポートに出力される．
(4) 外部からのストローブ($\overline{\mathrm{ASTB}}$)信号がHレベルに戻ると，その立ち上がりでレディ(ARDY)がLレベルに戻る．このとき，ポートの割り込みが有効な状態ならば，チャネルAから割り込みが発生する．

▶ PIO入力

(5) 外部からデータをチャネルAのポート上に乗せた状態で，ストローブ($\overline{\mathrm{BSTB}}$)信号がLレベルになる．
(6) チャネルAのデータがPIO内部にラッチされる．
(7) 外部からのストローブ($\overline{\mathrm{BSTB}}$)信号がHレベルに戻ると，その立ち上がりでレディ(BRDY)がLレベルになる．このとき，ポートの割り込みが有効な状態ならば，チャネルBから割り込みが発生する．
(8) CPUがポートのデータを読み出すと，レディ(BRDY)がHレベルに戻る．

以上のようにZ80PIOのモード2は，A/B両チャネルのレディ(ARDY/BRDY)とストローブ($\overline{\mathrm{ASTB}}$/$\overline{\mathrm{BSTB}}$)を使用します．

またチャネルAがモード2のとき，ポートBはモード3でしか動作できません．

モード2の動作タイミングを図6に示します．

● **Z80PIOモード3**(ビット・モード)

モード3はビット・モードという名前がついているように，ポートのビットごとに入力と出力の設定が可能で，モード0～2のような，レディ/ストローブなどのハンドシェイク線を使用しないため，いつでもポートの入出力ができるモードです．

ビットごとに入出力を設定できるといっても，Z80のI/O命令は8ビットごとにしか入出力できません．このときポートへのリード命令に対して，入力に指定されたビットはPIOのポートの状態を入力しますが，出力と指定されたビットについては，PIO内部の出力ラッチのデータが読み出され，合成されて1バイトのデータとしてCPUに送られます．

したがって，ポートの全ビットを出力と指定したと

きに，ポートからデータをリードすると，以前に出力したデータがそのまま入力(リード・バック)されます．

またポートが入力に設定されているとき，$\overline{RD}$信号の立ち下がりでポートの入力状態がPIO内部にラッチされるので，いつ変化するかわからない信号でも安定して読み込めます(図7)．

またモード3には，各信号線のビットの状態により割り込みを発生させることができます．つまりハンドシェイク線を使えなくても，例えばビット0とビット2が共にHレベルになったら割り込みを発生させるという指定が可能になります(図8)．

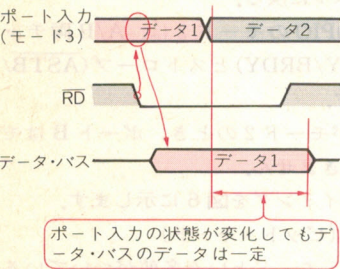
Z80PIOのプログラミング

● PIOのアクセス

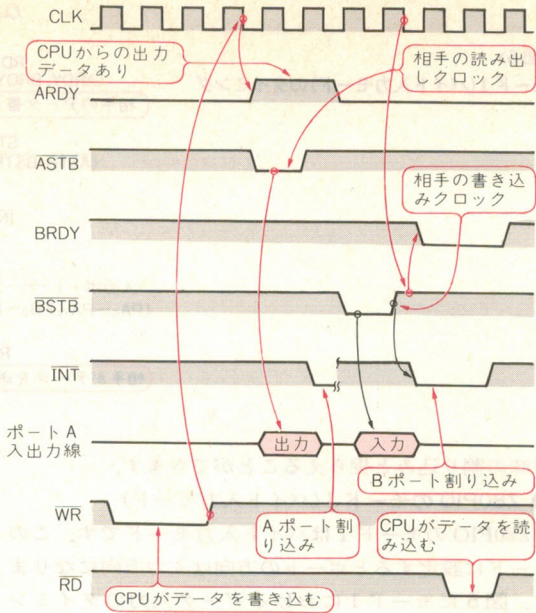
Z80PIOは，チャネルAとチャネルBの二つのチャネルがあり，それぞれにデータ・ポートとコントロール・ポートをもっています(表2)．

ここでCPUが，チャネルAのコントロール・ポートに対して書き込みをすると，チャネルAのモード設定や割り込みベクタの設定などができます．またチャネルBのデータ・ポートから読み込み動作を行うと，チャネルBが入力に設定されていればチャネルBのポートの状態が，出力に設定されていれば書き込んだデータがリード・バックして読み出されます．

図9にコントロール・ワードのフォーマットを示します．

〈図7〉モード3のビット入力時

〈表2〉Z80PIOの各ポートの選択

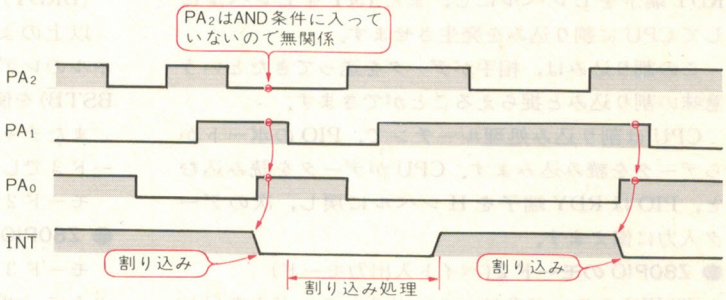
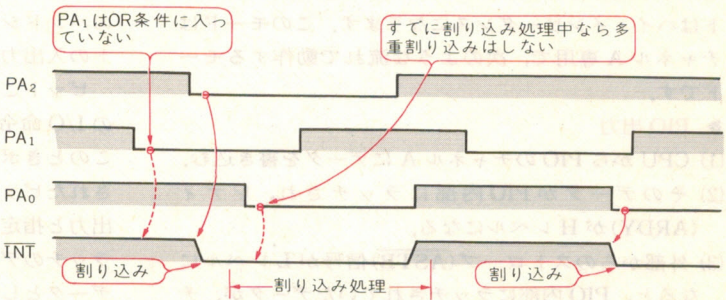
B/$\overline{A}$	C/$\overline{D}$	選択されるポート
"L"	"L"	チャネルAデータ・ポート
"L"	"H"	チャネルAコントロール・ポート
"H"	"L"	チャネルBデータ・ポート
"H"	"H"	チャネルBコントロール・ポート

〈図6〉モード2(バイト入出力モード)のタイミング

● モード0〜2初期化プログラム

Z80PIOはZ80周辺LSIとしては比較的単純なものですが，それでもモード設定のプログラミングには気を付けなければならないことがあります．

図10(a)にモード0〜2における初期化プログラムの流れを示します．

初期化プログラムで注意することは，初期化途中で

〈図8〉モード3の割り込みのAND，OR条件

(a) ビット0とビット1が"H"でAND条件のとき

(b) ビット2とビット0が"L"でOR条件のとき

〈図 9〉 Z80PIO のコントロール・ワードのフォーマットと初期化の流れ

(a) モード・ワード

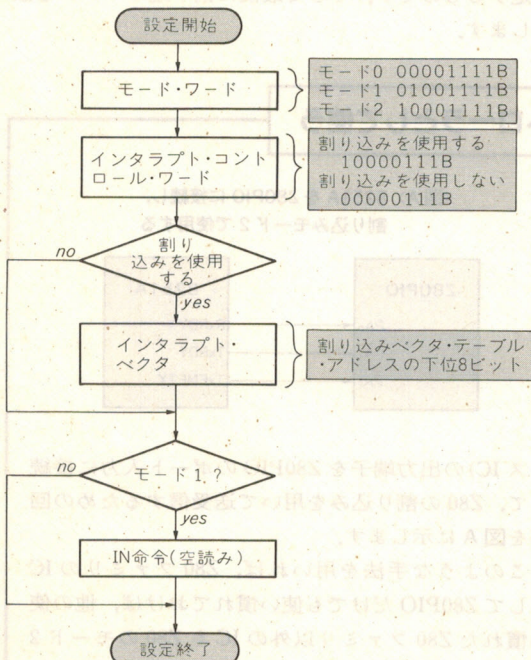
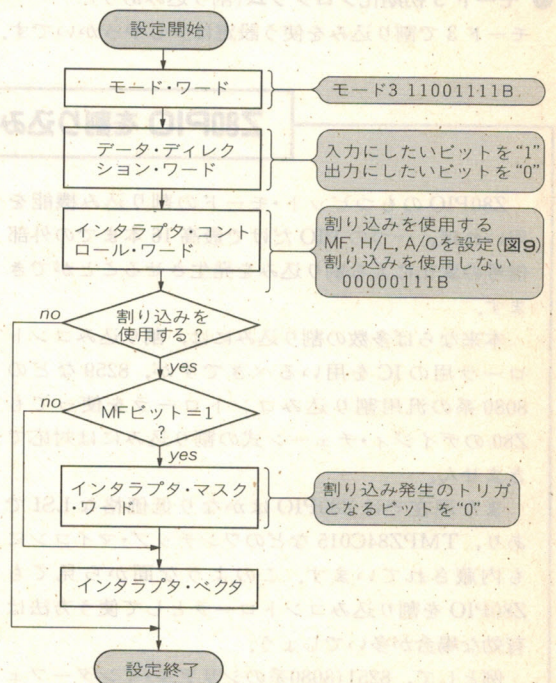
(b) データ・ディレクション・ワード

(c) インタラプト・コントロール・ワード(その1)

(d) インタラプト・コントロール・ワード(その2)

(e) インタラプト・マスク・ワード

(f) 割り込みベクタのフォーマット

〈図 10〉 PIO の各モードの初期化プログラムの流れ

(a) モード0, モード1, モード2における初期化プログラムの流れ

(b) モード3の初期化プログラムの流れ

割り込みが発生しないように，初期化の先頭で割り込み受け付けを禁止する点です．

またモード1の入力モードでは，初期化しただけではRDY端子がHレベルにならず，外部からデータを入力することができません．モード1ではIN命令を入れてRDY端子をHレベルにします．このIN命令のことを空読みとも呼びます．読み込んだデータそのものに意味はないので，データは無視してかまいません．

● **モード3初期化プログラム**(割り込みなし)

いちばん使用頻度の多いと思われる，単純なビット入出力モードであるモード3です．図10(b)に初期化プログラムの流れを示します．モード3では，ビットごとに入出力の設定ができたので，モード・ワードを書き込んだあとにディレクション・ワードが必要になります．そしてインタラプト・コントロール・ワードの最上位ビットを“0”にして書き込めば，割り込みを使わないモード3の設定が完了します．

モード3で割り込みを使わないといっても，モード初期化の途中で割り込みが発生しないように，割り込み受け付けは禁止します．これは，たとえ割り込みを使わなくても，3バイトのデータを書き込まないとモード3の初期化が終了しないからです．このとき，例えば2バイト目を書き込んだとき割り込みが発生し，その割り込みルーチンでもPIOに対してアクセスがあった場合，書き込みシーケンスの順番がおかしくなってしまい，正常に初期化されません．

● **モード3初期化プログラム**(割り込みあり)

モード3で割り込みを使う設定は少しやっかいです．割り込みを使う，使わないの設定は，インタラプト・コントロール・ワードのビット7で指定します．これを“1”にすると割り込みを使用します．

モード0やモード1では，RDY端子や$\overline{STB}$端子を制御することで割り込みを発生しましたが，モード3ではこれらの端子は使用しません．よって割り込みのトリガには，データ・ポート8ビットの状態変化をトリガにします．

そのための指定がビット4～ビット6の指定です．ビット4はMF(マスク・フォローズ)ビットで，このビットを1にすると，次にコントロール・ポートに書き込むデータはインタラプト・マスク・ワードであることを示します．

ビット5は，指定したビットがHレベルになったときに割り込みを発生させるのか，Lレベルになったときに割り込みを発生させるのかの指定です．

ビット6は，インタラプト・マスク・ワードで指定した割り込み発生の対象となるビットが複数の場合，すべての端子がビット5で指定したレベルになったら割り込みを発生させるのか，どれか一つでも指定したレベルになったら割り込みを発生させるのかを指定するビットです．

次は，インタラプト・コントロール・ワードのビット4のMFビットを1にした場合は，インタラプト・マスク・ワードを書き込みます．

これはデータ・ディレクション・ワードで入力に設定したビットのうち，割り込み発生の対象するビットを指定するものです．そして最後に割り込みベクタを設定します．

Z80PIOを割り込みコントローラとして使う

Z80PIOのもつビット・モードの割り込み機能を用いれば，一つのPIOだけで最高16本までの外部信号の変化による割り込みを発生させることができます．

本来ならば多数の割り込みには，割り込みコントローラ用のICを用いるべきですが，8259などの8080系の汎用割り込みコントローラを使ってもZ80のデイジィ・チェーン式の割り込みには対応できません．

また現在ではZ80PIOはかなり低価格なLSIであり，TMPZ84C015などのワンチップ・マイコンにも内蔵されています．このような面から見てもZ80PIOを割り込みコントローラとして使う方法は有効な場合が多いでしょう．

例として，8251(8080系のシリアル・インターフェース IC)の出力端子をZ80PIOのポート入力に接続して，Z80の割り込みを用いて送受信するための回路を図Aに示します．

〈図A〉8251AをZ80PIOに接続し，割り込みモード2で使用する

このような手法を用いれば，Z80ファミリのICとしてZ80PIOだけでも使い慣れておけば，他の使い慣れたZ80ファミリ以外のICもZ80のモード2割り込みで簡単に利用できるようになります．

Z80PIO の使用上の注意点

Z80PIO には，普段使っているときには気が付かないような問題など，注意すべき点がいくつかあります．そのあたりをちょっと整理しておきます．

● 電源 OFF 時/初期化時の端子の状態の変化

PIO はパワー ON(リセット)時にはチャネル A，B ともに入力に設定されます．ということは，外部からみればすべてハイ・インピーダンス状態になるということです．

通常はこのポートの端子にはプルアップ抵抗を接続し，これによってハイ・インピーダンス状態は H レベルになります．

ここまでは大した問題ではないのですが，つぎに CPU のプログラムが PIO の初期化を開始してモード設定をするときに，出力と設定されたポートの端子はモード設定後にはすべて L レベルを出力する点に注意します．

当然，その後に出力データとして “1” を書き込めば出力端子は H レベルになりますが，モード設定直後から “1” を書き込むまでの間は，どうしても L レベルが出力されてしまいます．

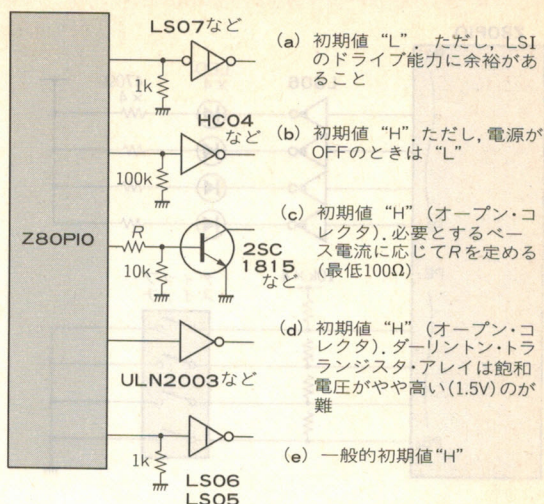
これは，短くすれば数 μs 程度の一般には短いと考えられる時間かもしれませんが，場合によっては 0.1 μs でも L レベルが出ると困ることも多くあります．

この問題には図 11 に示すような回路で対処する方法があります．これには(a)から(e)まで五つの方式を示しています．図(a)は，ポートからバッファ IC(LS07)を接続していますが，これにプルダウン抵抗を付加することで初期値は L レベルとなります．

図(b)は，初期値 H レベルを得るためにバッファではなくインバータを入れています．しかしこのインバータの出力回路の構成から，電源が入っていないと L レベルを出力してしまうので，これではまだ問題が残ります．

図(c)は，トランジスタを用いてオープン・コレクタで出力しています．これならば初期状態で H レベルが出力され，さらに電源 OFF 時もトランジスタは OFF なので，コレクタの先に接続された回路には影響を与えません．

図(d)も同様の考え方で，ダーリントン・トランジスタ・アレイによるものです．比較的大電流のドライブ向きです．ただし一般にダーリントン・トランジスタ・アレイは LS-TTL ほど高速ではありませんし，飽和電圧がやや高い(約 1.5 V)ことなど多少注意すべき点もあります．とはいっても ULN2003/2083 などは還流用などにも使えるダイオードも内蔵していますし，リレーなどを直接駆動するならばこの方法はかなり有効だと思います．

〈図 11〉 停電時/初期化時に不必要な信号を出さないために

最後に図(e)に最も一般的と思われる方法を示します．基本的に図(c)と同様の考え方のもので，オープン・コレクタの TTL を使っています．まとまったポート出力がある場合にはこのように TTL を使ったほうがスマートになる場合が多いと思います．

● モード 2 で，チャネル B がビット・モードのとき

モード 2(バイト入出力モード)で，チャネル A，B の割り込みルーチンで送受信をしているとき，チャネル B でモード 3 のビット・モードの割り込みを使用することはできません．

これは，ポート B のストローブ入力による割り込みとビット・モードの割り込みとを区別できない場合があるからです．

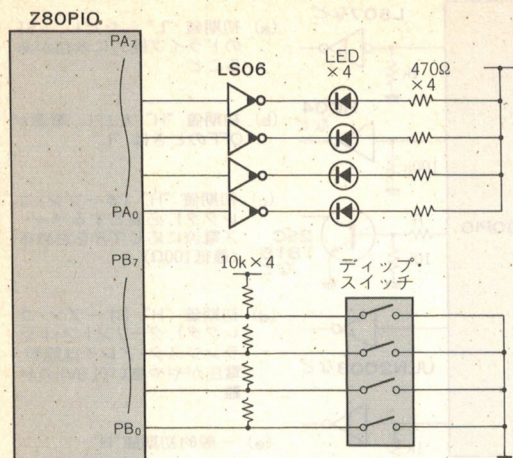
コラム(p.98)に示すように，Z80CPU に Z80 ファミリではない LSI を接続し，割り込みで使用するときなど，割り込みコントローラのない Z80 では PIO などの未使用ピンを割り込み発生源に使用するというテクニックがあります．

モード 2 のとき，チャネル B をビット・モードの割り込みで使いたいときは，モード 2 の入力によるチャネル B の割り込みと，ビット・モードでの割り込みを，きちんと区別できるような回路を設計する必要があります．

● モード 3(ビット・モード)の割り込みはエッジ・センス

ビット・モードの割り込みは H レベルや L レベルで発生するのではなく，L レベルから H レベルへの立ち上がりか，あるいは H レベルから L レベルへの立ち下がりで発生します．

つまり，立ち上がりエッジで割り込みするようにプログラムしてある場合は，いったん L レベルから H

〈図 12〉ディップ・スイッチ入力と LED 出力の回路

レベルに立ち上がったときに 1 回だけ CPU に割り込みがかかります．その後，再び L レベルに戻ってから，また H レベルに立ち上がらない限り CPU に割り込みはかかりません．

● モード 3（ビット・モード）の OR 条件で割り込みを使用する場合

Z80PIO をモード 3 の OR 条件で割り込みを使用する場合，ちょっと面倒な問題があります．それは，OR 条件になっているビットの中でどのビットの変化によって，割り込みが発生したのかを判定するための完全な情報を Z80PIO が返してくれないことです．

判定はポートのデータを読んでビットの状態を見れば良いのですが，例えばある二つのビットの，どちらかが H レベルになれば割り込みがかかるというようにプログラムしてある場合，どちらのビットも“1”（つまり H レベル）だったときなどです．どれが変化のあったビットなのかを確実に調べることは，割り込み先のプログラムだけでは不可能です．

このような場合，完全に判別できるように独立した割り込みベクタを発生できるように Z80 ファミリを 1 個追加するなど，システムを変更するのが理想的といえば理想的ですが…．筆者ならこうするという方法を次に示します．

▶ Z80CTC など別の割り込みに利用できる入力端子を利用する

Z80CTC には 4 チャネルのカウンタ入力があり，これらを用いることで 1 個の Z80CTC につき，4 入力までの割り込みを正確に発生することができます（詳しくは第 8 章を参照）．

▶ プログラムで一定周期ごとにポートの入力信号を監視する

Z80PIO の割り込みは使用せず，Z80CTC などのタイマ割り込みを一定周期で発生させ，そこで定期的に

〈リスト 1〉図 12 の回路用のテスト・プログラム例

```
;...... 基本定数 ......................................
;    動作環境に応じて修正
PIOAD   equ     1Ch     ;ＰＩＯポートＡデータ
PIOAC   equ     PIOAD+1 ;ＰＩＯポートＡコントロール
PIOBD   equ     PIOAD+2 ;ＰＩＯポートＢデータ
PIOBC   equ     PIOAD+3 ;ＰＩＯポートＢコントロール

;...... 初期化 .........................................
SwLed:
        ;...... ＰＩＯポートＡの初期化
        ld      a, 11001111b    ;モード3に設定
        out     (PIOAC), a
        ld      a, 00000000b    ;全ビット出力
        out     (PIOAC), a
        ld      a, 00000111b    ;割り込みは使用しない
        out     (PIOAC), a
        ;...... ＰＩＯポートＢの初期化
        ld      a, 11001111b    ;モード3に設定
        out     (PIOBC), a
        ld      a, 11111111b    ;全ビット入力
        out     (PIOBC), a
        ld      a, 00000111b    ;割り込みは使用しない
        out     (PIOBC), a

;...... メイン・プログラム ..........................
swled_loop:
        ;...... ディップ・スイッチ（ポートB）より入力
        in      a, (PIOBD)
        ;...... ＬＥＤ（ポートA）へ出力
        out     (PIOAD), a
        ;...... 繰り返し
        jr      swled_loop
```

PIO の端子をチェックするということです．ただし，当然タイマ割り込みによるプログラム処理のオーバ・ヘッドが多少あります．また最大の問題は信号の変化から判定までの時間の遅れが，タイマ割り込みの周期の時間だけ遅れることです．

▶ チャネル A/B とも，ビット・モードで割り込みを使用

これは両チャネルの入力端子はともに同様の接続をして，チャネル A は H レベル条件（立ち上がり）で割り込みをかけ，チャネル B は反対に L レベル条件（立ち下がり）で割り込みをかけるようにする方法です．

これならば，Z80CTC などを使わずに済みます．しかしポートを余計に使用することや，プログラム処理のオーバ・ヘッドを考えるとあまり現実的な方法ではないでしょう．

Z80PIO の応用例

● ディップ・スイッチ入力と LED 出力

まずは，4 ビットのディップ・スイッチ入力と 4 ビットの LED 出力を行う例から示します．回路例を図 12 に，プログラム例をリスト 1 に示します．

動作は単純にチャネル B のディップ・スイッチから入力したデータをチャネル A に出力するだけのものです．チャネル A，B ともに下位 4 ビットだけを使用していますが，この例では上位 4 ビットは未使用なので，プログラムでは特に意識せずに 8 ビット入力してそのまま 8 ビットを出力しています．

Z80PIO はモード 3（ビット・モード）で使用します．

割り込みなどは一切使用しません．

Z80PIO の使い方としては，このビット・モードが最も多いのではないかと思います．Z80PIO の基本中の基本です．

●セントロニクス準拠プリンタ・インターフェース(出力)

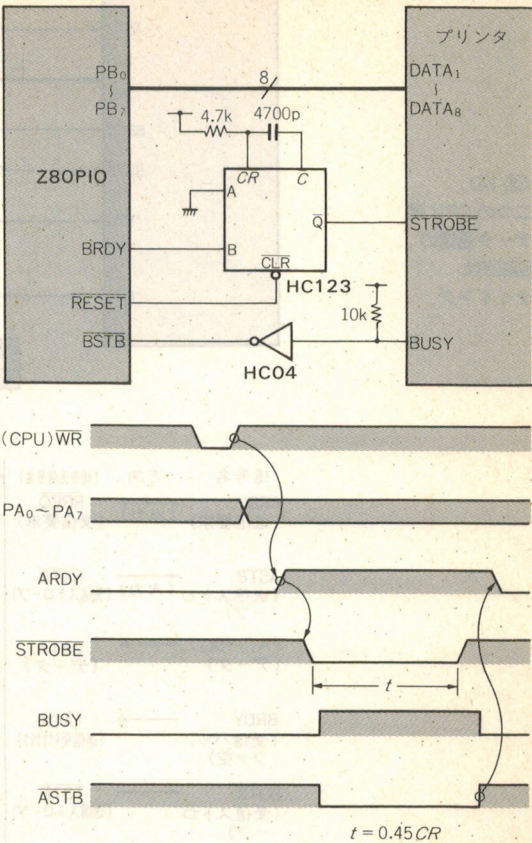
Z80PIO はチャネル A をモード 0(出力モード)で使用します．チャネル B はこの例では未使用です．割り込みはチャネル A で使用しています．

回路例を図 13 に，プログラム例をリスト 2 に示します．

CPU がデータを PIO に書き込むと，まず ARDY 端子が H レベルになります．これをトリガとしてワンショット・マルチバイブレータを使ってプリンタに書き込みクロック(ストローブ)を出力します．

プリンタ側では書き込みクロックが入力されると BUSY 端子を H レベルにし，1 文字受信処理の終了で L レベルに戻します．これが PIO の $\overline{\mathrm{ASTB}}$ に入力されているので，割り込みが発生し，次の印字データの出力となるわけです．

このプログラムを実行すると，プリンタに“Hello

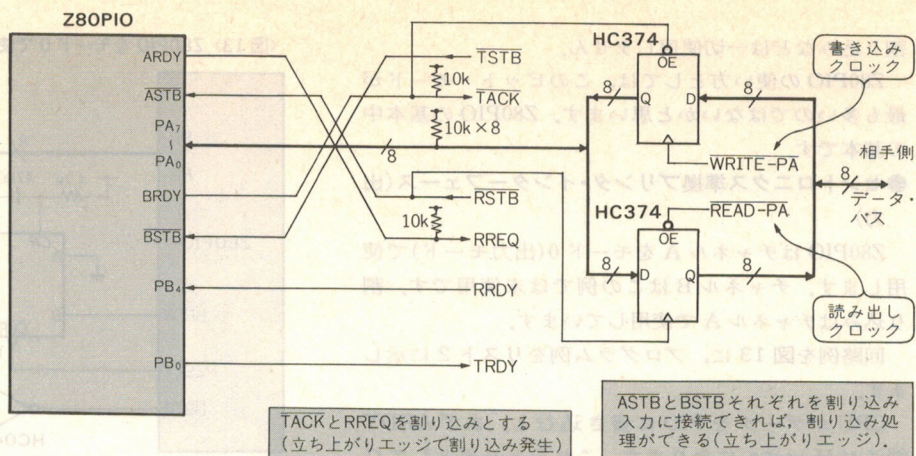
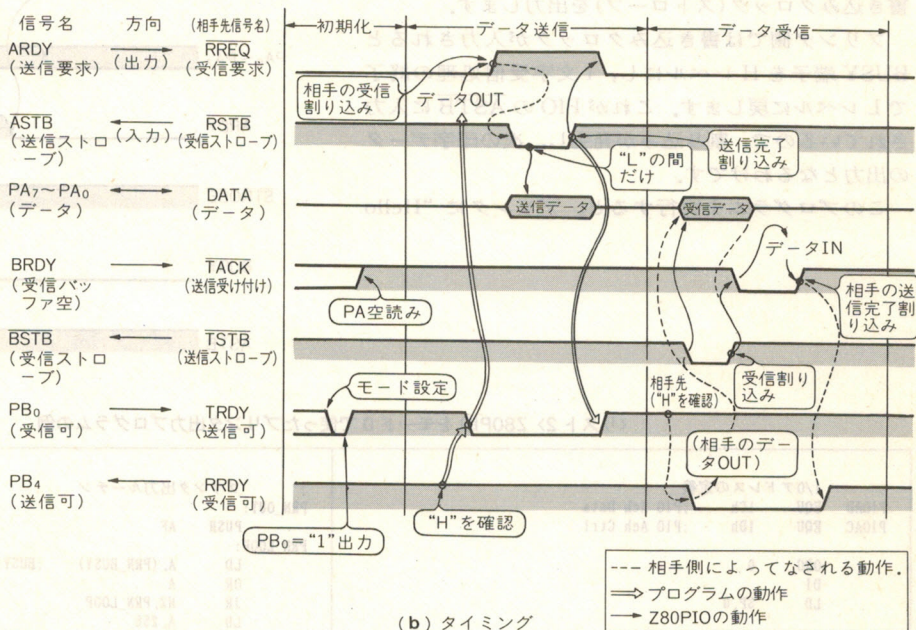
〈図 13〉Z80PIO をモード 0 で使ったプリンタ出力回路の例

〈リスト 2〉Z80PIO をモード 0 で使ったプリンタ出力プログラムの例

```
;       I/Oアドレスの定義
PIOAD   EQU     1Ch      ;PIO Ach Data
PIOAC   EQU     1Dh      ;PIO Ach Ctrl

        ORG     0
        DI
        LD      SP,0
;       PIO初期化
        DI                          ;割り込みモード設定
        IM      2
        LD      A,HIGH PIO_INT
        LD      I,A

        LD      A,00001111B         ;PIOモード0
        OUT     (PIOAC),A
        LD      A,10000111B         ;割り込み使用する
        OUT     (PIOAC),A
        LD      A,LOW PIO_INT       ;割り込みベクタ設定
        OUT     (PIOAC),A

        XOR     A                   ;ワークエリア初期化
        LD      (PRN_BUSY),A

        EI
        LD      HL,PRN_DATA
LOOP:   LD      A,(HL)              ;文字データ
        INC     HL
        OR      A
        JR      Z,PRN_END
        CALL    PRN_OUT             ;プリンタ出力
        JR      LOOP
PRN_END:
        DI
        HALT                        ;CPU停止
```

```
;       プリンタ出力ルーチン
PRN_OUT:
        PUSH    AF
PRN_LOOP:
        LD      A,(PRN_BUSY)        ;BUSYフラグ チェック
        OR      A
        JR      NZ,PRN_LOOP
        LD      A,255
        LD      (PRN_BUSY),A        ;BUSYフラグ セット
        POP     AF                  ;印字文字データ
        OUT     (PIOAD),A
        RET

;       PIO送信割り込みルーチン
PIO_ACH:
        PUSH    AF
        XOR     A
        LD      (PRN_BUSY),A        ;BUSYフラグ クリア
        POP     AF
        EI
        RETI

;       割り込みベクタ・テーブル
        ORG     ($+1) AND 0FFFEH
PIO_INT:
        DEFW    PIO_ACH

PRN_DATA:                           ;テスト印字データ
        DEFB    "CQ! CQ!",13,10,0

        ORG     8000H               ;RAM領域
PRN_BUSY:                           ;BUSYフラグ
        DEFS    0

        END
```

〈図 14〉
二つの CPU 間
データ通信の
回路例と
タイミング

(a) 回路例

(b) タイミング

オリジナル+α の Z80PIO

シャープ製の CMOS 版 Z80PIO(LH5081L)は，オリジナルのザイログ製 Z80PIO にはない，PIO のモードや $\overline{\text{STB}}$ 入力端子や RDY 出力端子の状態を読み出す機能が拡張されています．オリジナルの Z80PIO では，各チャネルのコントロール・ポートは書き込み専用で，読み出しができませんが，シャープ製の Z80PIO ではどちらのチャネルのコントロール・ポートでも両方のチャネルの状態を読み出せます(図 B)．これを使うとモード 3 で使用する場合，$\overline{\text{ASTB}}$ と $\overline{\text{BSTB}}$ の入力が 2 ビット増えることになります．

〈図 B〉ステータス情報ワード

D7	D6	D5	D4	D3	D2	D1	D0
AM_1	AM_0	ARDY	$\overline{\text{A STB}}$	BM_1	BM_0	B RDY	$\overline{\text{B STB}}$

Aポート・モード設定（D7～D6）　Bポート・モード設定（D3～D2）

M_1	M_0	モード
0	0	モード0:出力モード
0	1	モード1:入力モード
1	0	モード2:双方向モード(Aポートのみ)
1	1	モード3:ビット・モード

〈リスト 3〉 Z80PIO をモード 2 で使った CPU 間通信のプログラム例

```
;......  基本定数 ...............................................
;        動作環境に応じて修正
INTVEC  equu    0F000h          ;スタック&割り込みベクタ開始アドレス
INTPIO  equ     INTVEC+00h      ;PIO-A～ 割り込みベクタ開始アドレス
PIOAD   equ     1Ch             ;ＰＩＯポートＡデータ
PIOAC   equ     PIOAD+1         ;ＰＩＯポートＡコントロール
PIOBD   equ     PIOAD+2         ;ＰＩＯポートＢデータ
PIOBC   equ     PIOAD+3         ;ＰＩＯポートＢコントロール
;......  プログラム ...........................................
TwoCpuTest:
        ;......  初期化 (スタック&割り込み設定)
        di                      ;ベクタ書き換え等のため割り込み禁止
        ld      sp, INTVEC      ;スタック・ポインタを初期化
        ld      a, high INTVEC  ;
        ld      i, a
        im2
        ;......  ＣＰＵ間通信のための初期化
        ld      ix, INTPIO
        call    CmmInit
        ei                      ;割り込み許可が必要

        ;......  ＣＰＵ間通信テスト (Z80PIO側)
        ;  00h～FFhを順に送信して受信のくりかえし
        ld      a, 00h
tm20:    push   af
         call   CmmOut          ;送信
         call   CmmIn           ;受信
         pop    af
        inc     a
        jr tm20

;======  ＣＰＵ間通信制御モジュール ==============================
;------  ＣＰＵ間通信のための初期化 ------------------------------
CmmInit:
        ;......  送信/受信の割り込み先をベクタに書き込む
        ld      de, txint
        ld      (ix+0), e       ;PIO-A 割り込みベクタ内へ
        ld      (ix+1), d       ; txint アドレスを書き込む
        ld      de, rxint
        ld      (ix+2), e       ;PIO-B 割り込みベクタ内へ
        ld      (ix+3), d       ; rxint アドレスを書き込む
        ;......  ＰＩＯポートＡの初期化
        ld      a, 10001111b    ;モード２に設定
        out     (PIOAC), a
        ld      a, 10000111b    ;割り込みを使用 (ポートＡ:送信)
        out     (PIOAC), a
        push    ix
        pop     de              ;de = ix
        ld      a, e
        out     (PIOAC), a      ;ポートＡ割り込みベクトタ
        ;......  ＰＩＯポートＢの初期化
        ld      a, 11001111b    ;モード３に設定
        out     (PIOBC), a
        ld      a, 11110000b    ;ビット7～4:入力, ビット3～0:出力
        out     (PIOBC), a
        ld      a, 10110111b    ;割り込みを使用 (ポートＢ:受信)
        out     (PIOBC), a
        ld      a, 11111111b    ;ビット割り込みは使用しない(できない)
        out     (PIOBC), a
        inc     de
        inc     de
        ld      a, e
        out     (PIOBC), a      ;ポートＢ割り込みベクトタ
        ;......
        in      a, (PIOAD)      ;BRDY=H にするために
                                ;  一度データを空読みする
        xor     a               ;受信未処理
        ld      (rxflg), a      ;  フラグ = 0 (処理済)
        ld      a, 00000001b    ;PB0=1 (受信可)
        out     (PIOBD), a
        ret

;------ １バイト送信 ---------------------------------------------
CmmOut:
        ld      b, a            ;b= データ
        ;......  送信完了(PB0=1)かつ送信可(PB4=1)まで待つ
co10:   in      a, (PIOBD)
        bit     0, a            ;PB0=1 なら送信完了
        jr      z, co10
        bit     4, a            ;PB4=1 なら送信可
        jr      z, co10
        ;......  データ送信
        res     0, a            ;PB0=0 (受信不可)
        out     (PIOBD), a
        ld      a, b            ;a= データ
        out     (PIOAD), a
        ret

;------ １バイト受信 ---------------------------------------------
CmmIn:
        ;......  受信未処理フラグ (rxflg) がセットされるまで待つ
        ;         (受信割り込み待ち)
        ld      hl, rxflg
        xor     a               ;a=0
ci10:   or      (hl)
        jr      z, ci10
        ;......  受信データ(rxdata)取得して rxflg=0 にクリア
        ld      (hl), a         ;rxflg=0
        ld      a, (rxdata)     ;a= データ
        ret

;......  送信完了割り込みルーチン ...........................
txint:  push    af
        in      a, (PIOBD)
        set     0, a            ;PB0=0 (受信可)
        out     (PIOBD), a
        pop     af
        ei
        reti

;......  受信割り込みルーチン ...............................
rxint:  push    af
        ld      a, 1            ;rxflg=1
        ld      (rxflg), a
        ;
        in      a, (PIOAD)      ;データ受信
        ld      (rxdata), a     ;rxdata = 受信データ
        pop     af
        ei
        reti

        org     8000h
;......  モジュール用変数 ...................................
rxflg:  defs    1               ;受信未処理フラグ (0:処理済)
rxdata: defs    1               ;受信データ
```

Printer！”と1行だけ印字します．このプログラムは最初の1文字目の出力以降は，必要なときだけPIO割り込みで出力するようになっているので，印刷処理に費やされる時間の無駄が最小限に抑えられています．

またリストをみるとわかりますが，PIOの初期化ルーチン PrnInit を呼び出すときに，IX レジスタに割り込みベクタ・テーブルのアドレスを入れて呼び出しています．そして RAM 領域に割り込みベクタを設定しています．これも初期化のテクニックの一つです．

● 二つの CPU 間でのデータ通信

Z80PIO はモード 2(双方向モード)で使用します．割り込みは出力にチャネル A，入力にチャネル B と両方のチャネルを使用します．

ブロック図を図 14 に，プログラム例をリスト 3 に示します．

一般的には片側だけ Z80CPU で，もう片側はまったく別の CPU や DSP などの場合が多いと思います．そのような場合でも，8 ビットのデータ・バスと書き込みクロックと読み出しクロックが取れれば，ほとんどの CPU でインターフェースがとれるでしょう．当然ハンドシェイクのタイミングなどは注意してプログラムする必要があります．

ここにあげた回路は，いずれも距離が十数 cm 程度のごく短い間の通信用です．数十 cm のケーブルなどを用いて距離をのばしたい場合は，バッファを付加するなどの対処は必要です．

◆参考文献◆

(1) VOLUME 1 DATABOOK, MICROPROCESSORS AND PERIPHERRALS, ZILOG.
(2) Z80 Family User's Manual, ZILOG.

第8章

時間を計ったりクロック数を数える

Z80CTCの使い方

野口智樹

Z80CTCの概要

● Z80CTCとは

Z80CTC(Counter Timer Circuit)は4個の8ビット・カウンタからなるLSIです．システム・クロックを基準にして一定時間ごとに割り込みをかけたり，外部からのクロック数をカウントしたりできます．

Z80CTCは，次のような特徴があります．

- 4個の各チャネルはそれぞれ独立してカウンタまたはタイマ・モードで動作できる
- 各チャネルはシステム・クロックを1/16または1/256するプリスケーラを内蔵(タイマ・モードでのみ有効)
- Z80CPUのモード2割り込みに対応(割り込み優先順位はチャネル0が最高，チャネル3が最低)
- 各チャネルのゼロ/タイム・アウト出力は，ダーリントン・トランジスタをドライブ可能
- カウンタ・モードまたはタイマ・モードいずれの場合にも，CPUからカウンタの内容を読み出すことが可能
- タイマ・モード時のタイマ動作の起動用トリガの極性をプログラム可能
- カウンタ・モード時のカウント動作のトリガの極性をプログラム可能

● 内部ブロック図，ピン配置

Z80CTCのブロック図を図1に，ピン配置を図2に，図3と表1に電気的特性やタイミングを示します．

CPUとの接続についてはすでに説明しているので省略します(第4章参照)．

外部との入出力に使われる端子は，外部クロックまたはトリガ入力であるCLK/TRG_0～CLK/TRG_3と，カウント終了出力であるZC/TO_0～ZC/TO_2となります．

ピン配置をみるとわかりますが，Z80CTCのチャネル3には，ZC/TO出力がありません．これはピン数の制限によるものと思われます．また44ピンであるはずのQFPについても，チャネル3のZC/TO出力はありません．

Z80CTCの各種モード

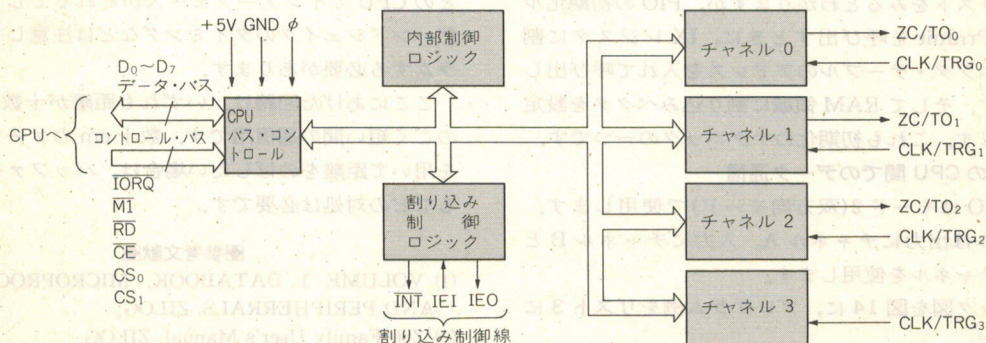
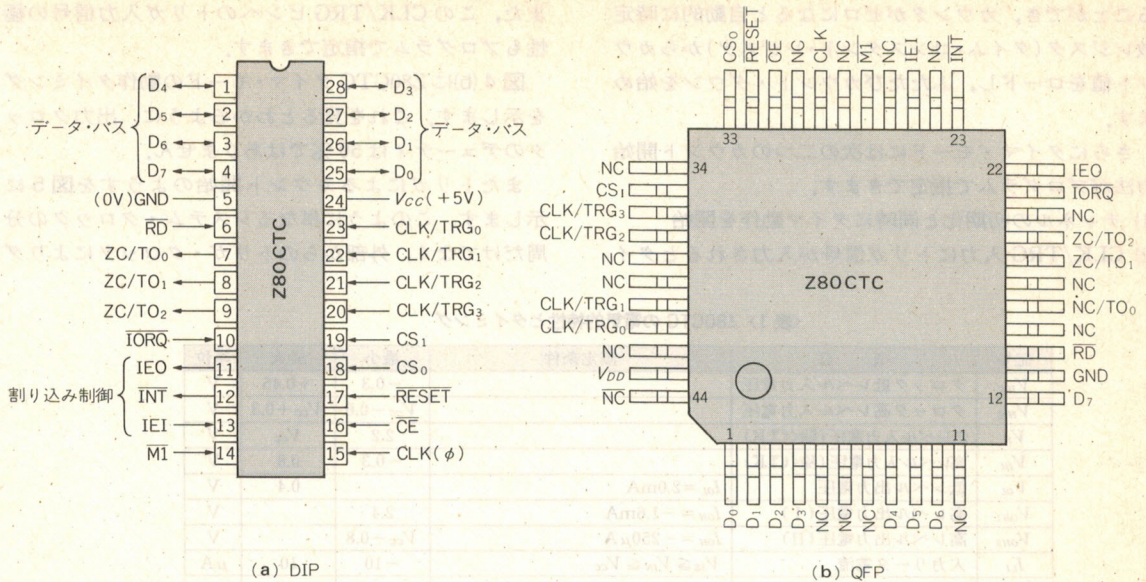
Z80CTCには大きく分けて，システム・クロックを基準クロックとして動作するタイマ・モードと，外部クロックを基準に動作するカウンタ・モードの二つがあります．

● Z80CTCカウンタ・モードの動作

カウンタ・モードでは，CTCはCLK/TRG入力信号のエッジによりダウン・カウント動作をします．エ

〈図1〉Z80CTCの構成

〈図2〉 Z80CTCのピン配置

(a) DIP　　　(b) QFP

ッジの極性(立ち上がりか立ち下がり)はプログラムで指定できます．

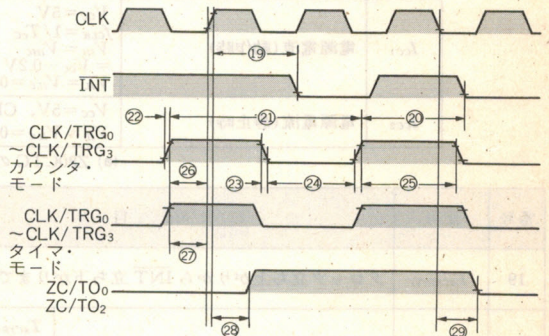
Z80CTCは，プログラムで指定された極性のエッジを検出すると，カウント値を−1します(ダウン・カウント)．

ここでカウント値が0になると，該当するチャネルのZC/TO出力ピンが1.5システム・クロックの間だけHレベルになります．このとき，CPUにゼロ・カウントの意味をもった割り込みをかけることができます．

なお，カウント可能な外部クロックは，Z80CTC内部がハードウェア的に8ビットのカウンタなので，2の8乗，つまり256までです．これを越えるカウントをしたい場合には，ゼロ・カウント割り込みなどを用いてソフト的に桁上げしたり，二つのカウンタをカスケード接続します．

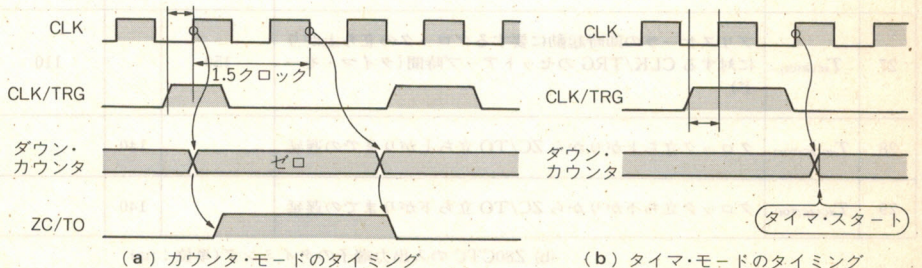
また外部クロック周波数は，システム・クロック周波数の1/2以下でないと，正常にカウントできません．

図4(a)にZ80CTCカウンタ・モードの動作タイミングを示します．

〈図3〉 Z80CTCのタイミング図

● Z80CTCタイマ・モードの動作

タイマ・モードでは，システム・クロックをまずプリスケーラに入力し分周します．そしてこの出力をカウント・ダウンします．プリスケーラは1/16または1/256を選択できるので，システム・クロックをϕとすると，1/16の場合は16ϕ〜4096ϕで，1/256の場合は256ϕ〜65536ϕの範囲でカウント可能です．

〈図4〉 Z80CTCの動作のようす

(a) カウンタ・モードのタイミング　　(b) タイマ・モードのタイミング

またカウントがゼロになった時点で割り込みをかけることができ，カウンタがゼロになると自動的に時定数レジスタ(タイム・コンスタント・レジスタ)からカウント値をロードし，ふたたびカウント・ダウンを始めます．

さらにタイマ・モードには次の二つのカウント開始方法がプログラムで指定できます．

(1) チャネルの初期化と同時にタイマ動作を開始

(2) CLK/TRG 入力にトリガ信号が入力されるとタイマ動作を開始

また，この CLK/TRG ピンへのトリガ入力信号の極性もプログラムで指定できます．

図 4 (b)に Z80CTC タイマ・モードの動作タイミングを示します．これを見るとわかるように，出力クロックのデューティは 50 %ではありません．

またトリガによるカウント開始のようすを図 5 に示します．このように単なるシステム・クロックの分周だけでなく，外部からのトリガ・クロックによりダ

〈表 1〉 Z80CTC の電気的特性とタイミング

記号	項　目	測定条件		最小	最大	単位
V_{ILC}	クロック低レベル入力電圧			−0.3	+0.45	V
V_{IHC}	クロック高レベル入力電圧			$V_{CC}-0.6$	$V_{CC}+0.3$	V
V_{IL}	低レベル入力電圧(除くCLK)			2.2	V_{CC}	V
V_{IH}	高レベル入力電圧(除くCLK)			−0.3	0.8	V
V_{OL}	低レベル出力電圧	$I_{OL}=2.0\text{mA}$			0.4	V
V_{OH1}	高レベル出力電圧(I)	$I_{OH}=-1.6\text{mA}$		2.4		V
V_{OH2}	高レベル出力電圧(II)	$I_{OH}=-250\mu\text{A}$		$V_{CC}-0.8$		V
I_{LI}	入力リーク電流	$V_{SS} \leqq V_{IN} \leqq V_{CC}$		−10	10	μA
I_{LO}	フローティング時 3 ステート出力電流	$V_{SS}+0.4 \leqq V_{OUT} \leqq V_{CC}$		−10	10	μA
I_{OHD}	ダーリントン駆動電流	$V_{OH}=1.5\text{V}$，$R_{EXT}=1.1\text{k}\Omega$ ZC/TO_0～ZC/TO_2に適用		−1.5	−5.0	mA
I_{CC1}	電源電流(動作時)	$V_{CC}=5\text{V}$ $f_{CLK}=1/T_{CC}$ $V_{IH}=V_{IHC}$ $=V_{CC}-0.2\text{V}$ $V_{IL}=V_{ILC}=0.2$	6MHz 版		8	mA
			8MHz 版		10	
			10MHz 版		12	
I_{CC2}	電源電流(静止時)	$V_{CC}=5\text{V}$，$\text{CLK}=V_{CC}$ $V_{IL}=V_{ILC}=0.2\text{V}$			10	μA

(a) Z80CTC の電気的特性

番号	記号	項　目		6MHz 版		8MHz 版		10MHz 版	
				最小	最大	最小	最大	最小	最大
19	$T_{dC(INT)}$	クロック立ち上がりから $\overline{\text{INT}}$ 立ち下がりまでの遅延		$T_{CC}+120$		$T_{CC}+100$		$T_{CC}+80$	
20	T_{dCLK}	CLK/TRG 立ち上がりから $\overline{\text{INT}}$ 立ち下がりまでの遅延(カウンタ・モード)	$T_{sCTR(C)}$を満足する場合	19+26		19+26		19+26	
			$T_{sCTR(C)}$を満足しない場合	1+19+26		1+19+26		1+19+26	
21	T_{cCTR}	CLK/TRG 周期		$2T_{CC}$		$2T_{CC}$		$2T_{CC}$	
22	T_{rCTR}	CLK/TRG 立ち上がり時間			40		30		30
23	T_{fCTR}	CLK/TRG 立ち下がり時間			40		30		30
24	T_{wCTRl}	CLK/TRG 低レベル・パルス幅		120		90		90	
25	T_{wCTRh}	CLK/TRG 高レベル・パルス幅		120		90		90	
26	$T_{sCTR(Cs)}$	即時カウントに要するクロックの立ち上がりに対する CLK/TRG のセットアップ時間(カウンタ・モード)		150		110		90	
27	$T_{sCTR(Ct)}$	プリスケーラの即時起動に要するクロックの立ち上がりに対する CLK/TRG のセットアップ時間(タイマ・モード)		150		110		90	
28	$T_{dC(ZC/TOr)}$	クロック立ち上がりから ZC/TO 立ち上がりまでの遅延			140		100		80
29	$T_{dC(ZC/TOf)}$	クロック立ち下がりから ZC/TO 立ち下がりまでの遅延			140		100		180

(b) Z80CTC の入出力端子のタイミング(単位：ns)

ウン・カウントを開始することもできます.

Z80CTCのプログラミング

● Z80CTCへのアクセス

Z80CTCはCS_0とCS_1の2本のチャネル・セレクト端子があり，それぞれ表2のようにチャネル0〜チャネル3までを選択します.

チャネルを選択し書き込み動作をすると，そのチャネルのコントロール・ワードを設定することになります．また読み出し動作をすると，そのチャネルのカウンタ値を読み出します.

● Z80CTCのモード設定のプログラミング方法

図6にコントロール・ワードのフォーマットを示します．タイム・コンスタントとは，カウント・ダウンを開始するときの最初の値です.

図7にZ80CTCの初期化プログラミングの流れを示します．これは，Z80CTCのモード設定の一般的な流れなので，これにしたがったモード設定を行えば問題になることはないでしょう.

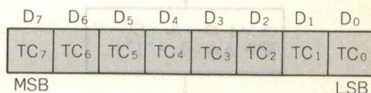
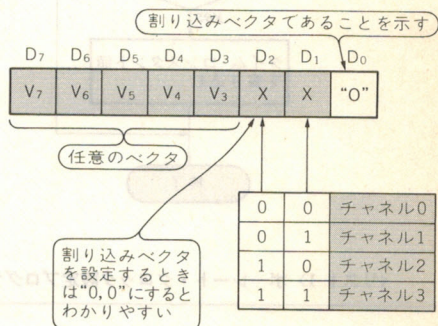
最初にいきなり割り込みベクタを書き込む点を不思議に思う方もいるでしょう．Z80の割り込みベクタは偶数番地を指定する必要があるので，必ず最下位ビットが“0”になります．CTCに対してのコントロー

〈図5〉外部トリガを使ったタイマ・モード

〈表2〉Z80CTCの各ポートの選択

CS_1	CS_0	セレクトされるチャネル	CPUの動作	CTCの動作
“L”	“L”	チャネル0	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み
“L”	“H”	チャネル1	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み
“H”	“L”	チャネル2	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み
“H”	“H”	チャネル3	リード	カウント値読み出し
			ライト	コントロール・ワード書き込み

〈図6〉Z80CTCのコントロール・ワード

D_2 ロード・タイム・コンスタント	D_1 リセット	動作
0	0	現在の動作を継続
0	1	現在の動作を停止
1	0	ダウン・カウンタの内容が0になった後，新しい時間定数がロードされる
1	1	現在の動作が停止し，ただちに新しい時間定数がロードされる

(a) チャネル・コントロール・ワード

(b) タイム・コンスタント・ワード

(c) 割り込みベクタのフォーマット(チャネル0のみ)

ル・ワードは最下位ビットを“1”にするので，区別がつくわけです．

ここでコントロール・ワードのリセット(D_1)について説明します．これはCTCのカウント動作を停止させるか，継続させるかの設定です．これをセットしていると，システム・クロックが入力されても外部からクロックが入力されても，カウント動作は止まったままになります．

またロード・タイム・コンスタント(D_2)ビットとの組み合わせでは，すでに設定してあるダウン・カウンタが0になってから，新しく設定したタイム・コンスタントがロードされ，カウントを始めるようになります(図8)．

● 割り込みベクタの設定

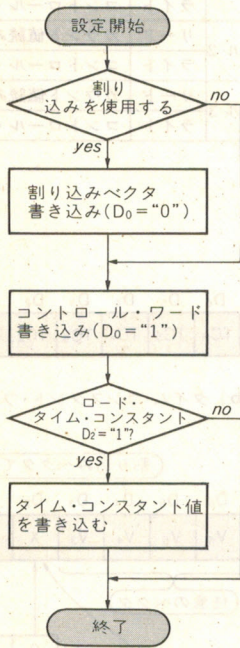
Z80CTCは割り込みベクタの設定をチャネル0に対してのみ行うことができます．チャネル0以外の割り込みベクタは，チャネル0に書かれたベクタのうち図6(c)に示すように，ビット2とビット1がチャネルごとに自動的に設定されます．

ただし，チャネル0に割り込みベクタとして設定された値に+2されたアドレスがチャネル1，+4されたアドレスがチャネル2…と考えるのは誤りです．

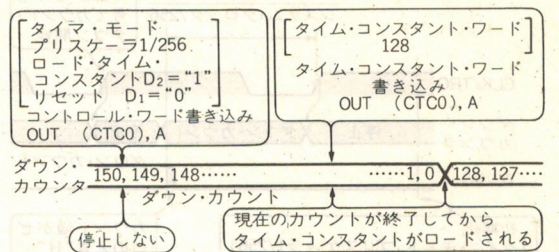
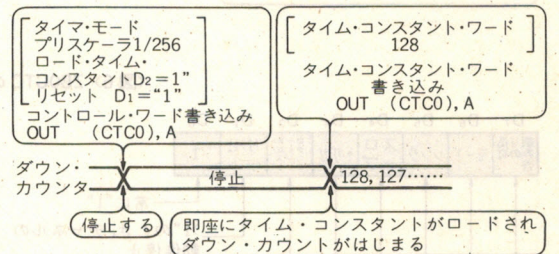
CTC内部では，ビット1とビット2にどんな値が入っていようとも，チャネル0なら“00”で，チャネル2なら“10”と置き換えてベクタを出力します．

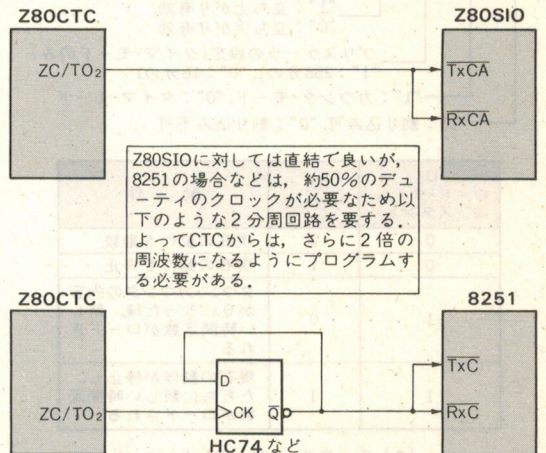
よってCTCの割り込みベクタ・テーブル・アドレスとしては，アドレスの下位3ビットが“0”になるアドレスを先頭にして，チャネル0から3までを順番に並べなければなりません．

〈図7〉 Z80CTCの初期化の流れ

設定開始
割り込みを使用する
yes
no
割り込みベクタ書き込み(D_0=“0”)
コントロール・ワード書き込み(D_0=“1”)
ロード・タイム・コンスタント D_2=“1”?
yes
no
タイム・コンスタント値を書き込む
終了

〈リスト1〉 ボーレート・クロック設定プログラム

```
;......  基本定数 ............................................
;     動作環境に応じて修正
CTC2    equu    12h                  ;ＣＴＣ Ch.2
BPSCLK  equ     4                    ;9600bps (CLK=9.8304MHz,SIO=X16)
;......  ボーレイト設定（ＣＴＣ初期化） ........................
        ld      a, 00000111b         ;割り込みなし、タイマ・モード、1/16
        out     (CTC2), a
        ld      a, BPSCLK            ;タイム・コンスタント
        out     (CTC2), a
```

〈図8〉 コントロール・ワードのD_2とD_1の関係

(a) リセットD_2=“0”のとき

(b) リセットD_2=“1”のとき

〈図9〉 シリアル通信用ボーレート・クロック生成の回路例

Z80CTCの応用例

● シリアル通信ボーレート・クロック生成

これはZ80CTCとしてはもっとも使用頻度が高い用途ではないでしょうか．回路例が図9で，プログラム例は**リスト1**です．

すでに説明したようにタイマ・モードであってもZ80CTCの出力はデューティ50％ではないので，8251などの送受信クロックとしてはそのままでは使えません．50％のデューティにするために，フリップフロップを一つ追加して2分周してから8251に接続します．もちろんこの場合には，Z80CTCの出力周波数を2倍にして考える必要があります．

Z80SIOを使う場合はそのままZC/TO出力を送受信クロック端子に接続してだいじょうぶです．

CTCの分周値の計算式は次式となります．

システム・クロック÷(ボーレート×SIO倍率)÷16

例えばシステム・クロック9.8304 MHz，プリスケーラ1/16，通信速度9600 bps，SIOのクロック倍率×16とした場合，

$$9830400\div(9600\times16)\div16=4$$

つまり，9.8304 MHz÷16÷4＝153600 Hzのクロックが出力されます．

● インターバル・タイマ

この例もCTCの代表的な使用例といえるものです．インターバル・タイマとは，一定周期ごとにかかるタイマ割り込みのことです．タイマ割り込みそのものにハードウェアはとくに必要ありません．**リスト2**にプログラム例を示します．

例えば1/60秒ごとにポートをチェックしたいなどのときに応用します．リスト中のタイマ割り込みルーチンに処理を埋め込めばよいだけです．

〈リスト2〉インターバル・タイマのプログラム例

```
;...... 基本定数 ..............................................
;    動作環境に応じて修正
INTVEC  equ     0F000h          ;スタック&割り込みベクタ開始アドレス
INTCTC  equ     INTVEC+10h      ;CTC Ch0～ 割り込みベクタ開始アドレス
CTC0    equ     10h             ;C T C  Ch.0
TINTCYC equ     64              ;タイマ割り込み周期の定数
;...... プログラム ............................................
        cseg
        ;...... 初期化 (スタック&割り込み設定)
        di                      ;ベクタ書き換え等のため割り込み禁止
        ld      sp, INTVEC      ;スタック・ポインタを初期化
        ld      a, high INTVEC  ;
        ld      i, a
        im2
        ;...... インターバル・タイマのための初期化
        ld      ix, INTCTC
        ld      iy, intsv
        ld      c, TINTCYC
        call    TimInit
        ei                      ;割り込み許可すること
        ;++++++ 動作テスト ++++++++++++++++++++++++++++++++++++
        ;       (メイン処理は、intsv: で行われる)
        jr      $               ;何もせず、単にくりかえし

        ;++++++ インターバル・タイマ割り込みルーチン +++++++++
        ;       割り込み毎にスイッチを一瞬ONしてOFF
intsv:  push    af
        ;
        ; ここに何らかのタイマ割り込みルーチンを書き込む
        ;
        pop     af
        ei
        reti

;====== インターバル・タイマ・モジュール =======================
;------ インターバル・タイマのための初期化 --------------------
TimInit:
        ;...... CTC0 設定
        push    ix
        pop     de              ;de = ix
        ld      a, e
        out     (CTC0), a       ;CTC 割り込みベクタ
        push    iy
        pop     de              ;de = iy
        ld      (ix+0), e       ;CTC Ch0 割り込みベクタ内へ
        ld      (ix+1), d       ; timint アドレスを書き込む
        ld      a, 10100111b    ;割り込みあり タイマ・モード 1/256
        out     (CTC0), a
        ld      a, c            ;タイム・コンスタント
        out     (CTC0), a
        ;......
        ret
```

● 回転数測定

Z80CTCは，ハードウェア的には8ビットのカウンタですが，最近では16ビットのカウンタがマイコン用カウンタICとしては主流のようです．とはいっても，8ビットで困ることはほとんどありません．それは，プログラムで適当に桁上げしてあげれば簡単に16ビットでも32ビットでも拡張できるからです．

Z80CTCを割り込みコントローラとして使う

Z80PIOの章でも少し触れましたが，Z80CTCの未使用カウンタを使って，Z80以外のLSIをZ80のモード2の割り込みで使うことができます．

Z80CTCには4チャネルのカウンタ入力があるので，これらを用いることで1個のZ80CTCにつき4入力までの割り込みを発生することができます．

Z80CTCはカウンタ・モードに設定し，タイム・コンスタント・レジスタに1を設定しておけば，割り込みエッジ入力後にカウント値が0になって，CPUに対して割り込みが発生します．

また都合の良いことに，Z80PIOと同じように割り込み入力信号のエッジをプログラムで指定することができます．

さらに二つ以上の割り込み入力信号が同時に入った場合なども，Z80CTCの各チャネルは独立して割り込みベクタを出力するので問題ありません．このような場合も，割り込みの優先順位はチャネル0が最優先となります．

〈リスト 3〉 回転数測定のプログラム例

```
;...... 基本定数 .......................................................
;    動作環境に応じて修正
INTVEC  equ     0F000h          ;スタック&割り込みベクタ開始アドレス
INTCTC  equ     INTVEC+10h      ;CTC Ch0～ 割り込みベクタ開始アドレス
CTC0    equ     10h             ;ＣＴＣ Ch.0
CTC1    equ     CTC0+1          ;ＣＴＣ Ch.1
T600HZ  equ     64              ;600Hz周期タイマ割り込みの定数
                                ;    (RpsInit: の説明を参照)
;...... プログラム .....................................................
RpsTest:
        ;...... 初期化 (スタック&割り込み設定)
        di                      ;ベクタ書き換え等のため割り込み禁止
        ld      sp, INTVEC      ;スタック・ポインタを初期化
        ld      a, high INTVEC  ;
        ld      i, a
        im2
        ;...... 回転数測定のための初期化
        ld      ix, INTCTC
        ld      c, T600HZ
        call    RpsInit
        ei                      ;
        ;...... 現在の回転数(rps)を表示
tst10:  call    RpsGet          ;回転数が hl に格納
        call    DispHL          ; hl を表示
        jr      tst10           ;くりかえし

;====== 回転数測定モジュール ==========================================
;...... モジュール用定数 ...............................................
mtime           equ     600     ;測定周期 [600=1秒:rps]
;------ 回転数測定のための初期化 ---------------------------------------
RpsInit:
        ;...... 変数初期化
        ld      hl, mtime
        ld      (rrem), hl
        ld      hl, 0
        ld      (rcnt), hl
        ld      (rps), hl
        ;...... CTC0 設定
        push    ix
        pop     de              ;de = ix
        ld      a, e
        out     (CTC0), a       ;CTC 割り込みベクタ・ワード
        ld      de, timint
        ld      (ix+0), e       ;CTC Ch0 割り込みベクタ内へ
        ld      (ix+1), d       ; timint アドレスを書き込む
        ld      a, 10100111b    ;割り込み有、タイマモード、1/256
        out     (CTC0), a
        ld      a, c            ;タイムコンスタント
        out     (CTC0), a
        ;...... CTC1 設定
        ld      a, 01000111b    ;割り込み無、カウンタモード、立ち下がり
        out     (CTC1), a
        ld      a, 0            ;タイムコンスタント
        out     (CTC1), a
        ;......
        ret

;------ 回転数測定 (タイマ割り込み) ------------------------------------
timint:
        push    af
        push    bc
        push    de
        push    hl
        ;......
        ld      de, (rrem)

        ld      a, d
        or      e               ;rrem=0 ならば
        jr      z, ti50         ;  回転数を決定
        dec     de              ;rrem ← rrem-1
        ld      (rrem), de
        ;...... カウント
        ld      hl, (rcnt)
        in      a, (CTC1)
        ld      b, a
        ld      a, (prvcnt)     ;1つ前のカウント－現在のカウント
        sub     b               ;  の結果を a に代入する
        ld      e, a
        ld      d, 0
        add     hl, de          ;カウント累計 (rcnt) に16bit加算
        ld      (rcnt), hl
        ld      a, b
        ld      (prvcnt), a     ;1つ前のカウント←現在のカウント
        jr      ti90
ti50:   ;...... rrem=0 になったので回転数を決定
        ld      hl, (rcnt)
        ld      (rps), hl       ;rcnt値がそのままrps値となる
        ld      hl, 0
        ld      (rcnt), hl
        ld      hl, mtime       ;次回の割り込みのために
        ld      (rrem), hl      ; rremを再設定
ti90:   ;......
        pop     hl
        pop     de
        pop     bc
        pop     af
        ei
        reti

;------ 回転数測定値取得 -----------------------------------------------
RpsGet:
        ld      hl, (rps)       ;現在の決定値をそのまま返す
        ret

;       org     8000            ;RAM 領域
;...... モジュール用変数 ...............................................
rrem:           defs    2       ;最終値決定までの割り込み回数残り
rcnt:           defs    2       ;現在までの回転カウント累計
rps:            defs    2       ;測定回転数 (最終値)
prvcnt:         defs    1       ;1つ前のＣＴＣカウント値
```

回路例を図 10 に，プログラム例をリスト 3 に示します．

このプログラムでは，チャネル 0 は一定周期のタイマ割り込みとして用い，チャネル 1 を回転クロック・カウンタとして用いています．

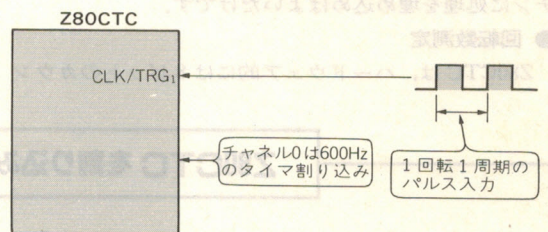
ポイントは，チャネル 1 のカウント値をチャネル 0 のタイマ割り込みごとに，16 ビットのカウント値に桁上げしていることです．こうすることで一定周期ごとに正確なカウント値を取り込むことができます．周期は 600 Hz (0.001666..秒) とプログラムしていますが，この間に 256 以上のカウント値が入力されると誤動作してしまうので，計測できる最高周波数が決まります．

〈図 10〉 回転数測定の回路例

◆参考文献◆

(1) VOLUME 1 DATABOOK, MICROPROCESSORS AND PERIPHERRALS, ZILOG.
(2) Z80 Family User's Manual, ZILOG.

第9章

シリアル通信の初期化から送受信プログラムまで

Z80SIOの使い方

高見　豊

Z80SIOの概要

● Z80SIOとは

Z80SIO(Serial Input Output)はZ80ファミリのシリアル通信コントローラLSIで，1パッケージに独立した2チャネルのUSART(Universal Synchronous Asynchronous Receiver and Transmitter：汎用同期非同期送受信器)を内蔵しています．また，SIOは非同期(調歩同期)から，BSCなどのキャラクタ同期，HDLCなどのフラグ同期まで対応できます．さらに送受信割り込みはもちろん，DCDやCTSなどのモデム・ステータスの変化によっても割り込みが可能なため，ポーリングによるソフトウェアの負担を最小限に抑えることができます．

SIOは，非同期通信で1.2 Mbps(クロック6 MHzの場合)の高速通信が実現できます．

またSIOと同じザイログ社の汎用(CPUを選ばない)USARTであるSCCはSIOと上位互換性があるため，SIOの使い方を知っていればSCCも難しくはありません．

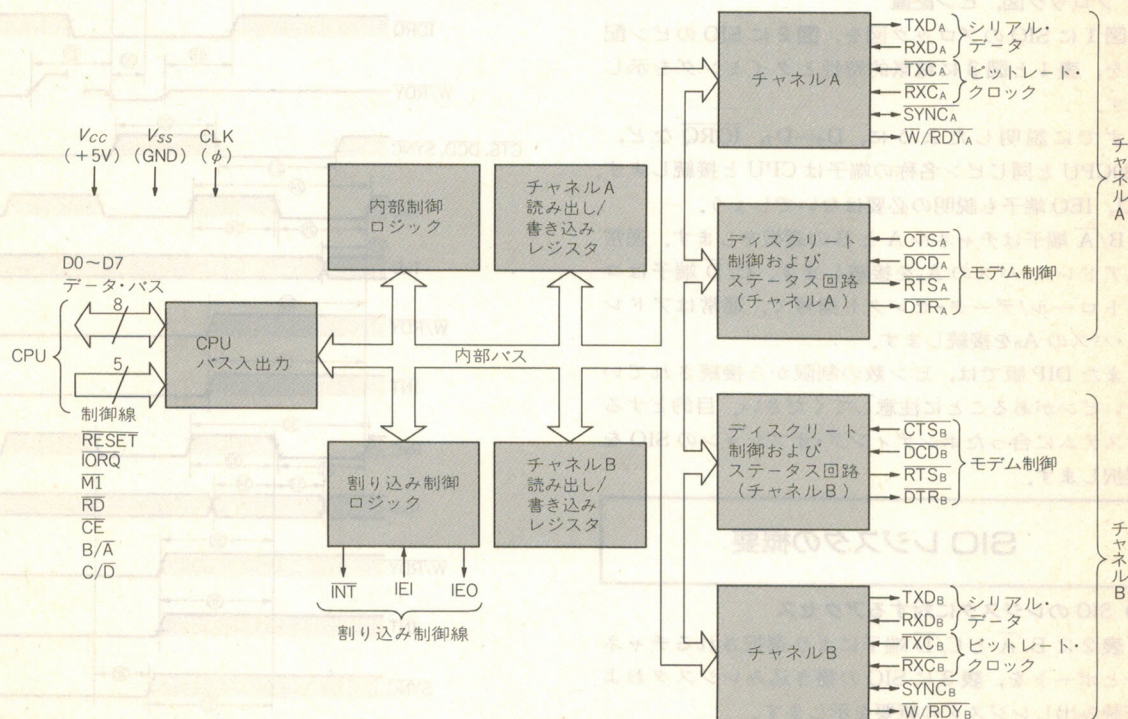
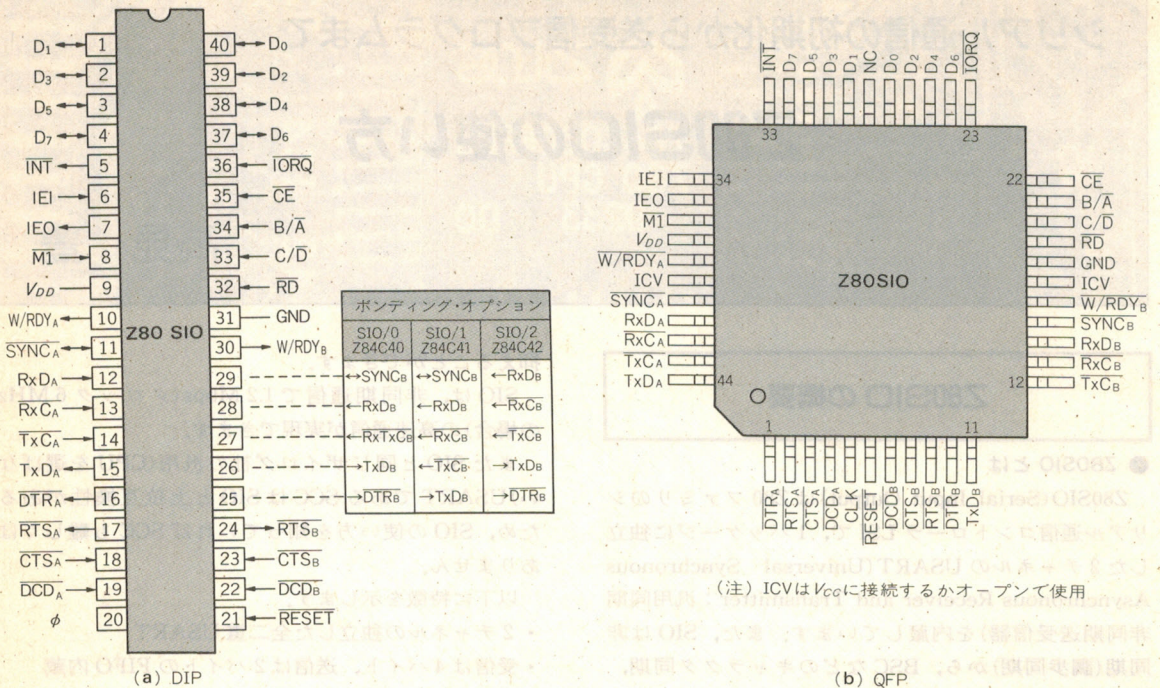
以下に特徴を示します．

- 2チャネルの独立した全二重USART
- 受信は4バイト，送信は2バイトのFIFO内蔵
- 1.2 Mbps(クロック6 MHz)までの転送レート対応
- Z80ファミリのデイジィ・チェーン割り込みをサポート
- 非同期，キャラクタ同期(BSC)，フラグ同期

〈図1〉Z80SIOのブロック図

〈図 2〉 Z80SIO のピン配置

ボンディング・オプション		
SIO/0 Z84C40	SIO/1 Z84C41	SIO/2 Z84C42
↔SYNCB	↔SYNCB	←RxDB
←RxDB	←RxDB	←$\overline{RxCB}$
←$\overline{RxTxCB}$	←$\overline{RxCB}$	←$\overline{TxCB}$
→TxDB	←$\overline{TxCB}$	→TxDB
→$\overline{DTRB}$	→TxDB	→$\overline{DTRB}$

(a) DIP

(注) ICVはV_{CC}に接続するかオープンで使用

(b) QFP

(HDLC)が可能

- CRC 生成，チェック機能内蔵
- HDLC/SDLC コンパチブル
- CCITT-X.25 コンパチブル

● **ブロック図，ピン配置**

図1にSIOのブロック図を，図2にSIOのピン配置を，表1と図3に電気的特性とタイミングを示します．

すでに説明したように，D_0～D_7，$\overline{IORQ}$など，Z80CPUと同じピン名称の端子はCPUと接続します．IEI，IEO端子も説明の必要はないでしょう．

B/$\overline{A}$端子はチャネルAとBの選択をします．通常はアドレス・バスのA_1を接続します．C/$\overline{D}$端子はコントロール/データ・セレクト信号で，通常はアドレス・バスのA_0を接続します．

またDIP版では，ピン数の制限から接続されていないピンがあることに注意してください．目的とするシステムに合ったボンディング・オプションのSIOを選択します．

SIO レジスタの概要

● **SIO のレジスタに対するアクセス**

表2にB/$\overline{A}$とC/$\overline{D}$端子により選択されるチャネルとポートを，表3にSIOの書き込みレジスタおよび読み出しレジスタの概要を示します．

〈図 3〉 Z80SIO のタイミング図

〈表 1〉 Z80SIO の電気的特性とタイミング

記号	項目	測定条件		最小	最大	単位
V_{ILC}	クロック低レベル入力電圧			−0.3	±0.45	V
V_{IHC}	クロック高レベル入力電圧			$V_{CC}-0.6$	$V_{CC}+0.3$	V
V_{IL}	低レベル入力電圧(除く CLK)			2.2	V_{CC}	V
V_{IH}	高レベル入力電圧(除く CLK)			−0.3	0.8	V
V_{OL}	低レベル出力電圧	$I_{OL}=2.0$ mA			0.4	V
V_{OH1}	高レベル出力電圧(I)	$I_{OH}=1.6$ mA		2.4		V
V_{OH2}	高レベル出力電圧(II)	$I_{OH}=-250\ \mu$A		$V_{CC}-0.8$		V
I_{LI}	入力リーク電流	$V_{SS}\leqq V_{IN}\leqq V_{CC}$		−10	10	μA
I_{LO}	出力リーク電流	$V_{SS}+0.4\leqq V_{OUT}\leqq V_{CC}$		−10	10	μA
$I_{L(SY)}$	$\overline{SYNC}$ 端子入力電流	$V_{SS}+0.4\leqq V_{IN}\leqq V_{CC}$		−40	10	μA
I_{CC1}	電源電流(動作時)	$V_{CC}=5$ V $f_{CLK}=1/T_{CC}$ $V_{IHC}=V_{IH}$ $=V_{CC}-0.2$ V $V_{ILC}=V_{IL}$ $=0.2$ V	6 MHz 版		10	mA
			8 MHz 版		12	
			10 MHz 版		15	
I_{CC2}	電源電流(静止時)	$V_{CC}=5$ V $V_{IH}=V_{IHC}=V_{CC}-0.2$ V $V_{IL}=V_{ILC}=0.2$ V			10	μA

(a) 電気的特性

番号	記号	項目	6 MHz 版		8 MHz 版		10 MHz 版	
			最小	最大	最小	最大	最小	最大
18	$T_{dIO(W/RWf)}$	$\overline{IORQ}$, $\overline{CE}$ の立ち下がりから $\overline{W/RDY}$ 立ち下がりまでの遅延(ウェイト・モード)		175		130		110
19	$T_{dC(W/RR)}$	クロックの立ち上がりから $\overline{W/RDY}$ 立ち下がりまでの遅延(レディ・モード)	100		95		85	
20	$T_{dC(W/RWz)}$	クロックの立ち下がりから $\overline{W/RDY}$ フロート状態までの遅延(ウェイト・モード)		110		90		80
22	T_{wPh}	高レベル・パルス幅	200		150		150	
23	T_{wPl}	低レベル・パルス幅	200		150		150	
24	T_{cTXC}	送信クロック周期	330		250		200	
25	T_{wTxCl}	低レベル送信クロック・パルス幅	100		85		80	
26	T_{wTxCh}	高レベル送信クロック・パルス幅	100		85		80	
27	$T_{dTxC(TxD)}$	$\overline{TxC}$ の立ち下がりから TxD 信号までの遅延(SIO クロック×1 モード)		220		160		120
28	$T_{dTxC(W/RRf)}$	$\overline{TxC}$ の立ち下がりから $\overline{W/RDY}$ 信号までの遅延(レディ・モード)	5	9	5	9	5	9
29	$T_{dTxC(INT)}$	$\overline{TxC}$ の立ち下がりから $\overline{INT}$ 立ち下がりまでの遅延	5	9	5	9	5	9
30	T_{cRxC}	受信クロック周期	330		250		200	
31	T_{wRxCl}	低レベル受信クロック・パルス幅	100		85		80	
32	T_{wRxCh}	高レベル受信クロック・パルス幅	100		85		80	
33	$T_{sRxD(RxC)}$	$\overline{RxC}$ 立ち上がりに対する RxD 信号セットアップ時間(SIO クロック×1 モード)	0		0		0	
34	$T_{hRxD(RxC)}$	$\overline{RxC}$ 立ち上がりに対する RxD 信号ホールド時間(SIO クロック×1 モード)	100		80		60	
35	$T_{dRxC(W/RRf)}$	$\overline{RxC}$ 立ち上がりから $\overline{W/RDY}$ 立ち下がりまでの遅延(レディ・モード)	10	13	10	13	10	13
36	$T_{dRxC(INT)}$	$\overline{RxC}$ の立ち上がりから $\overline{INT}$ 立ち下がりまでの遅延	10	13	10	13	10	13
37	$T_{dRxC(SYNC)}$	$\overline{RxC}$ 立ち上がりから $\overline{SYNC}$ 立ち下がりまでの遅延(出力モード)	4	7	4	7	4	7
38	$T_{sSYNC(RxC)}$	$\overline{RxC}$ の立ち上がりに対する $\overline{SYNC}$ 信号セットアップ時間(外部同期モード)	−100		−100		−100	

(b) Z80SIO の入出力端子のタイミング(単位：ns)

〈表2〉
SIOの各ポートの選択

B/$\overline{A}$	C/$\overline{D}$	セレクトされるポート	CPUの動作	SIO動作
"L"	"L"	チャネルAデータ・ポート	リード	受信データ
			ライト	送信データ
"L"	"H"	チャネルAコントロール・ポート	リード	リード・レジスタ(RR_0~RR_1)
			ライト	ライト・レジスタ(WR_0~WR_7)
"H"	"L"	チャネルBデータ・ポート	リード	受信データ
			ライト	送信データ
"H"	"H"	チャネルBコントロール・ポート	リード	リード・レジスタ(RR_0~RR_2)
			ライト	ライト・レジスタ(WR_0~WR_7)

〈表3〉Z80SIOのライト/リード・レジスタ構成

レジスタ名	記号	役目
ライト・レジスタ0	WR_0	コマンドおよびポインタ・ビット
ライト・レジスタ1	WR_1	割り込み制御および$\overline{WAIT}$/READYの制御
ライト・レジスタ2	WR_2	割り込みベクトル(チャネルBのみ)
ライト・レジスタ3	WR_3	レシーバ制御
ライト・レジスタ4	WR_4	コミュニケーション・モードの設定
ライト・レジスタ5	WR_5	トランスミッタ制御
ライト・レジスタ6	WR_6	シンク・キャラクタまたは従局アドレス(SDLC)
ライト・レジスタ7	WR_7	シンク・キャラクタまたはフラグ・パターン(HDLC)

(a) ライト・レジスタ

レジスタ名	記号	役目
リード・レジスタ0	RR_0	割り込み要因の識別
リード・レジスタ1	RR_1	エラー状態の識別およびレシデュー・コード
リード・レジスタ2	RR_2	割り込みベクトル(チャネルBのみ)

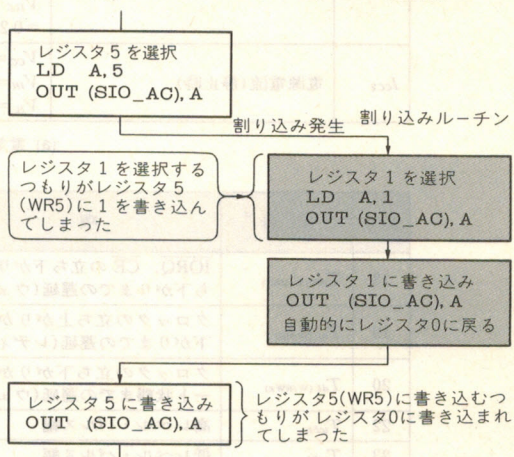
(b) リード・レジスタ

〈図4〉SIOのレジスタ・アクセスの注意点

これらのリード/ライト・レジスタは，一部例外を除きチャネルAおよびチャネルBそれぞれ独立して存在しています．これらのレジスタに対するアクセスは，各チャネルのコマンド・ポートに書き込み動作をすることでライト・レジスタへ書き込まれ，読み込み動作をすることでリード・レジスタより読み込まれます．

WR_0~WR_7，あるいはRR_0~RR_2まであるレジスタの切り替えは，レジスタ・ポインタを設定することでコントロール・ポートを一時的に切り替えるようにして使います．

リセット直後や読み書きシーケンスの終了状態では，コマンド・ポートに書き込みするとWR_0に書き込まれ，読み込みをするとRR_0の内容が読み込まれます．

ここでWR_0に01hを書き込むと，レジスタ・ポインタは1になり，コントロール・ポートはWR_1またはRR_1に切り替えられます．この次にコントロール・ポートに対して書き込み動作をするとWR_1に書き込まれます．一度コントロール・ポートにアクセスすると，またレジスタ・ポインタはクリアされ，コントロール・ポートはWR_0とRR_0に戻ります．

● レジスタ・ポインタの注意

いちどアクセスするだけで，レジスタ・ポインタはクリアされてしまうので，WR_2にデータを書き込んでRR_2のデータを読むという場合でも，再度レジスタ・ポインタを2に設定しなければなりません．

またレジスタ・ポインタ設定中に割り込みが発生し，その割り込みルーチンでもSIOのレジスタを操作していた場合，図4のように割り込み処理終了後のSIOのレジスタに対するアクセスでは，意図しないレジスタに対して書き込みをしてしまう場合があります．

よってレジスタ・ポインタを設定して目的のレジスタの読み書きが終了するまでは，割り込み受け付け禁止状態にしておきます．

また読み込み動作時はリード・レジスタを読み込む

ので，ライト・レジスタに書き込んだ値が後で必要になっても読み出すことはできません．書き込み専用レジスタです．必要ならばワーク・エリア上に保存しておきます．

● **プログラミング上の注意点**

SIOのプログラミングでとくに注意することは，書き込みレジスタ4(WR_4)を他のレジスタより先に設定しなければならないことです．WR_4は通信モードや送受信クロックの倍率を決定するレジスタで，これらの設定により送受信データやプログラミングが影響を受けるためです．特に同期通信では重要です．

また，SIOのレジスタにはチャンネルAにしかないものやチャンネルBにしかないものもあります．以下に各レジスタの詳細を示します．

書き込みレジスタの詳細

多機能さに比例して，レジスタの数も多いZ80SIOです．ここではレジスタの一覧をまとめて図5(pp. 116〜117)に示します．

● **書き込みレジスタ0(WR_0)**

レジスタ・ポインタと，基本動作コマンドの設定レジスタです．

▶ D_0〜D_2：レジスタ・ポインタ・ビット

これらのビットは，次のデータの書き込みまたは読み出しをするレジスタの番号を指定します．書き込みレジスタ0(WR_0)にOUT命令で出力した後に書き込み(OUT命令を実行)すると，そのデータはここで指定された番号の書き込みレジスタ(WR_0〜WR_7)に書き込まれます．また，出力した後に読み出し(IN命令を実行)すると，ここで指定された番号の読み出しレジスタから読み込みます．

▶ D_3〜D_5：基本コマンド・ビット

これらのビットはSIOの基本的な動作を指定するためのものです．

・000b：ノー・オペレーション

何もしないコマンドです．このコマンドは単にレジスタ・ポインタをセットしたい場合に使います．例えばコマンド・ポートに00000001bと書き込むと，通信動作に影響を与えることなく，レジスタ・ポインタを1(WR_1/RR_1)に設定します．

・001b：アボート発生

HDLCモードにおいて，アボート・シーケンス(7個の連続した“1”)を送信するためのコマンドです．データをどのように設定してもHDLCのゼロ・インサーション機能により“1”が5個連続すると自動的に“0”を挿入するので，このコマンドはアボートを発生する唯一の方法です．

・010b：外部/ステータス割り込みのリセット

外部/ステータス($\overline{CTS}$端子，$\overline{DCD}$端子，シンク/ハント，送信アンダーラン/EOM)が変化した場合に，SIOはRR_0をラッチします．よってRR_0で読み込むことのできる外部/ステータスは，その時点での最新の状態ではなく，最初にどれかが変化したときラッチされた状態となります．よって今の状態を知りたいとき，または外部/ステータス割り込み処理ルーチンの中で，次の変化で再度割り込みを発生させる場合はこのコマンドを実行しなければなりません．

・011b：チャネル・リセット

書き込まれたチャネルのすべての状態を$\overline{RESET}$端子によりリセットされたのと同様に初期化します．$\overline{RESET}$端子によるリセットと異なり，書き込まれたチャネルしかリセットしませんから，ソフトウェアによりSIOを完全にリセットしたい場合はチャネルAとチャネルBの両方に対してチャネル・リセット(18h)を書き込まなくてはなりません．

なお，チャネルAに対するチャネル・リセットは割り込み優先回路をもリセットします．

・100b：次のキャラクタ受信時の割り込みをイネーブル

このコマンドは，書き込みレジスタ1(WR_1)のビット3〜4で最初のキャラクタ受信時に割り込みが選択されている状態で，最初のキャラクタ受信の割り込みを処理した後に，次のキャラクタ受信で割り込みを発生させたい場合に実行します．詳しくは後述します．

・101b：保留中の送信割り込みをリセット

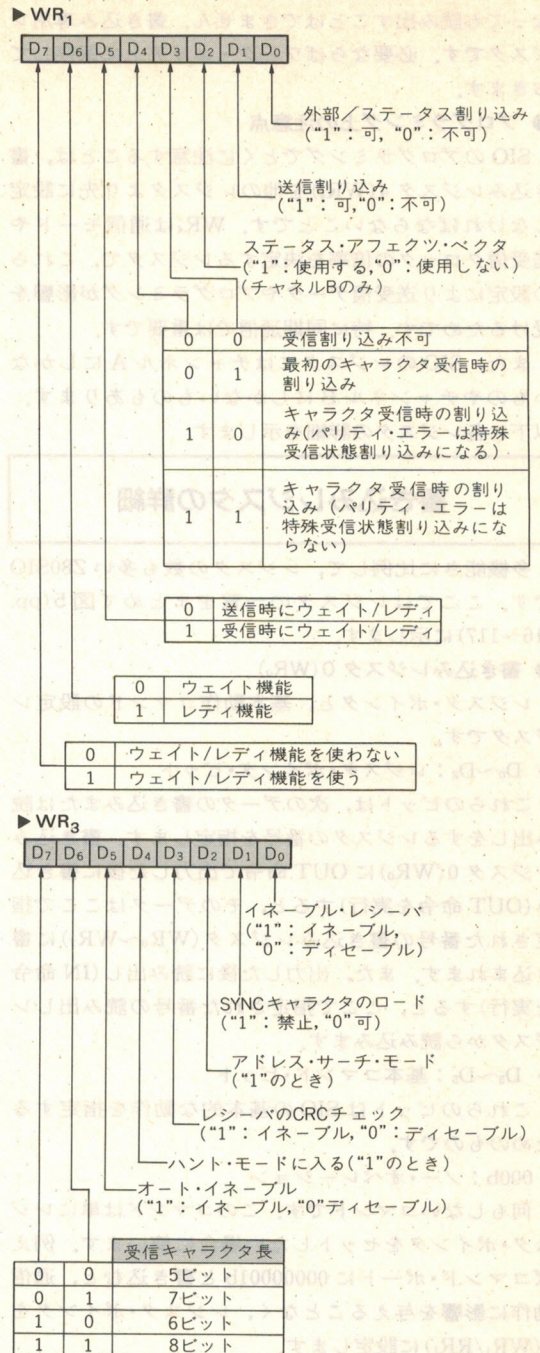
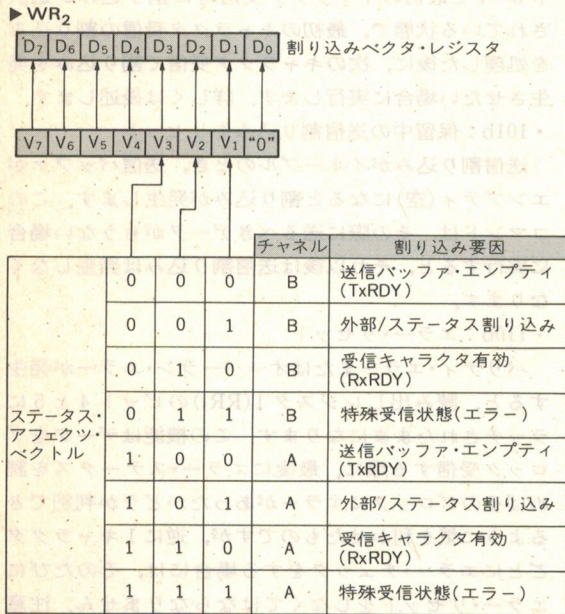
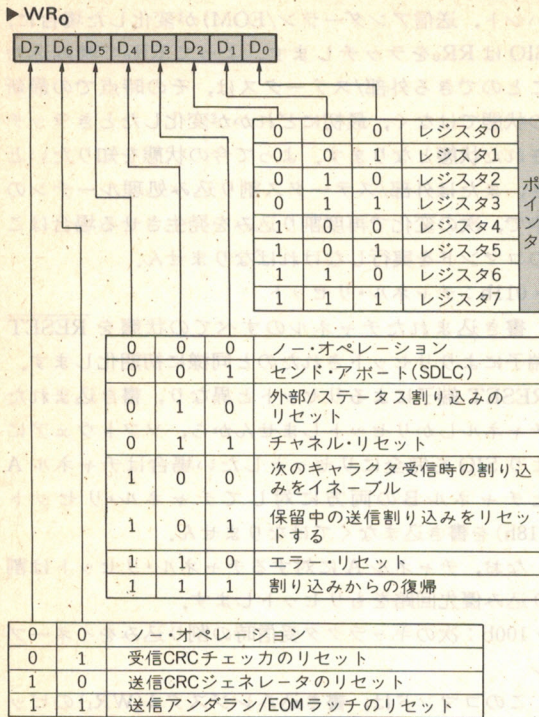
送信割り込みがイネーブルのとき，送信バッファがエンプティ(空)になると割り込みが発生します．このコマンドは，その際に送るべきデータがもうない場合に実行すると，それ以後は送信割り込みは発生しなくなります．

・110b：エラー・リセット

パリティ・エラーまたはオーバーラン・エラーが発生すると，読み出しレジスタ1(RR_1)のビット4と5にラッチされたままになります．この機能はデータをブロック受信する際に，最後にエラー・ステータスを読めばそのブロックにエラーがあったかどうか判別できるように気を利かせたものですが，逆に1キャラクタごとにエラー・チェックをする場合には，そのたびにエラー・リセットをしなくてはならなりません．注意が必要です．

・111b：割り込みからの復帰

このコマンドはZ80以外のCPUでZ80SIOを使うときに，RETI命令の実行により割り込みデイジィ・チェーンの復帰をする手段として用意されています．しかし実際にはZ80SIOをZ80以外のCPUに接続するには外部回路が大きくなりすぎます．ほぼ同機能のLSIであるμPD7201Aや72001A(どちらもNEC)な

〈図5〉Z80SIOのレジスター覧

どを使うのが現実的ですので，あまり使われません．

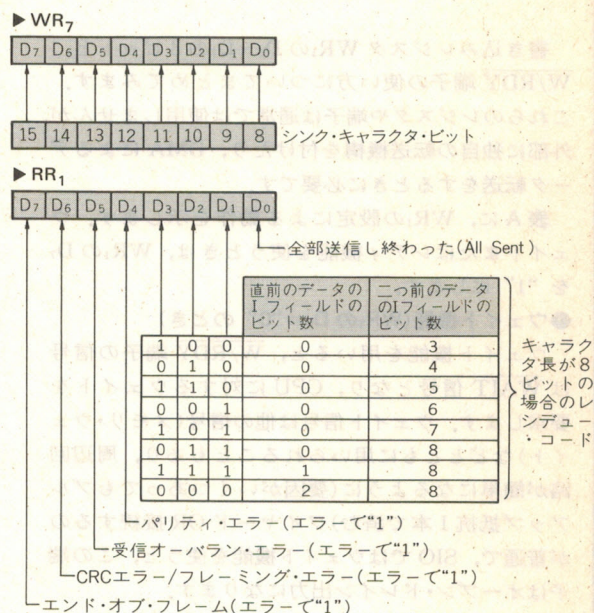
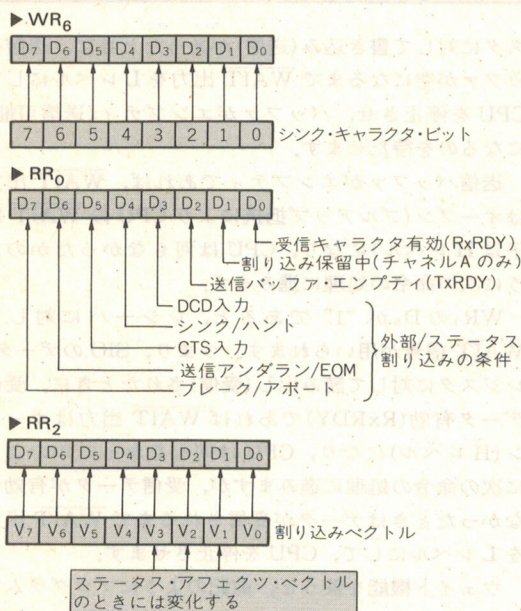
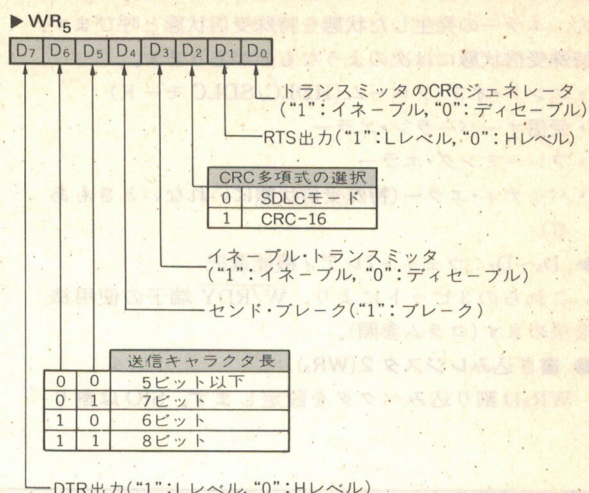
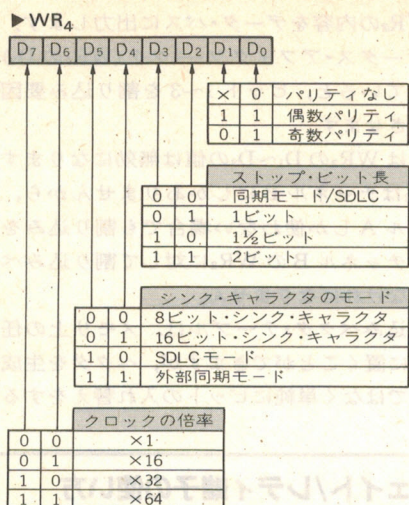
▶ D_6～D_7：CRCリセット・コマンド

この2ビットで受信CRCチェッカのリセット，送信CRCジェネレータのリセット，EOMのセット(HDLCモードでFCSおよび終了フラグを送信する)が行えます．

● 書き込みレジスタ1(WR_1)

WR_1は割り込みとウェイト/レディの設定をします．

▶ D_0：外部/ステータス割り込みイネーブル

このビットがセットされていると，非同期モードでブレーク信号(連続したスペース状態＝Lレベル)やHDLCモードでアボート・シーケンスを受信した場合，

$\overline{\mathrm{CTS}}$端子や$\overline{\mathrm{DCD}}$端子，SYNC端子の状態が変化した場合，送信アンダーラン/HDLCモードでのエンド・オブ・フレーム(EOM)で外部/ステータス割り込みを発生します．

▶ D₁：送信割り込みイネーブル

このビットがセットされていると，送信バッファがエンプティ(空)になると送信割り込みを発生します．

▶ D₂：ステータス・アフェクツ・ベクタ

このビットがセットされていると，割り込み発生時に割り込みチャネルや要因の種類によりWR_2に設定されている割り込みベクタ(下位8ビット)のD_1～D_3を変化させて，割り込みベクタとして出力します．この機能は割り込み要因によりベクタを変えることで，割り込み処理ルーチンの中で要因を判別する手間を省き，処理を高速に行うためのもので，CPUが割り込みモード2で使われている場合に有効です．

▶ D₃～D₄：受信割り込みモード

これらのビットは，受信割り込みのモードを決めます．

- 00b：受信割り込みを使用しない
- 01b：最初のキャラクタ受信時のみ割り込みを発生
- 10b：キャラクタ受信割り込み(パリティ・エラーは特殊受信状態割り込み)
- 11b：キャラクタ受信割り込み(パリティ・エラーは特殊受信状態割り込みにならない)

パリティ・エラーを除いて，基本的にはエラーが発

生したときは，キャラクタ受信割り込みは発生しません．エラーの発生した状態を特殊受信状態と呼びます．特殊受信状態には次のようなものがあります．

・エンド・オブ・フレーム(HDLC/SDLC モード)
・受信オーバーラン・エラー
・フレーミング・エラー
・パリティ・エラー(特殊受信状態にいれないときもある)

▶ D_5～D_7：ウェイト/レディ機能選択

これらの3ビットにより，$\overline{W/RDY}$ 端子の使用法を決めます(コラム参照)．

● 書き込みレジスタ 2(WR_2)

WR_2は割り込みベクタを設定します．SIO は割り込みを発生させ，CPU より割り込みベクタの要求があると，WR_2の内容をデータ・バスに出力します．このときステータス・アフェクツ・ベクタ(WR_1の D_2)がセットされていると，ビット 1～3 を割り込み要因に応じて変化させます．

この場合は WR_2の D_1～D_3の値は無効になります．また，WR_2はチャネル B にしかありませんから，たとえチャネル A しか使わない場合でも割り込みを使う場合は，チャネル B の WR_2に対して割り込みベクタを設定しなければなりません．

また割り込みベクタ・テーブルは，メモリ上の任意のアドレスに置くことができますが，ベクタを生成する際に加算ではなく単純にビットの入れ替えをするだ

ウェイト/レディ端子の使い方

書き込みレジスタ WR_1の D_5～D_7の 3 ビットと，$\overline{W/RDY}$ 端子の使い方についてまとめてみます．これらのレジスタや端子は通常では使用しませんが，外部に独自の転送機構を付けたり，DMA によるデータ転送をするときに必要です．

表 A に，WR_1の設定による動作を示します．ウェイトまたはレディ機能を使うときは，WR_1の D_7を“1”にします．

●ウェイト機能(WR_1の D_6=“0” のとき)

ウェイト機能を用いると，$\overline{W/RDY}$ 端子の信号は $\overline{WAIT}$ 信号となり，CPU に対するウェイトを要求します．ウェイト信号は他の信号(メモリ・ウェイト)などとともに用いられることもあり，周辺回路が簡単になるように(要因がいくつあってもプルアップ抵抗 1 本で済む)ワイヤード OR 接続するのが普通で，SIO ではウェイト機能を使うと，この端子はオープン・ドレイン出力になります．

ウェイト機能が指定されているとき，WR_1の D_5が“0”であるとトランスミッタに対して $\overline{WAIT}$ 信号が用いられます．つまり，SIO のデータ・レジスタに対して書き込み(送信)があったときに，送信バッファが空になるまで $\overline{WAIT}$ 出力を L レベルにして CPU を停止させ，バッファがエンプティ(送信可能)になるのを待たせます．

送信バッファがエンプティであれば，$\overline{WAIT}$ 出力はオープン(プルアップ抵抗により CPU の $\overline{WAIT}$ 端子は H レベル)になり，CPU は何もなかったかのように次の命令の処理に進みます．

WR_1の D_5が“1”であると，レシーバに対して $\overline{WAIT}$ 信号が用いられます．つまり，SIO のデータ・レジスタに対して読み出し(受信)されたときに，受信データ有効(RxRDY)であれば $\overline{WAIT}$ 出力はオープン(H レベル)になり，CPU は何もなかったかのように次の命令の処理に進みますが，受信データが有効でなかったときはデータが受信されるまで $\overline{WAIT}$ 出力を L レベルにして，CPU を停止させます．

ウェイト機能を使うと，前記のようにプログラムで送信バッファ・エンプティ(TxRDY)や受信データ有効(RxRDY)ビットをポーリング(“1” になるまで待つ)することなく，データの送信または受信ができま

〈表 A〉 ウェイト/レディ端子の使い方

WR_1			$\overline{W/RDY_A}$, $\overline{W/RDY_B}$の機能	$\overline{H/L}$ レベル	状態
D_7	D_6	D_5			
1	0	0	$\overline{WAIT}$ (TxRDY)	L	トランスミッタがバッファ・エンプティでない(送信できない)
				フローティング	トランスミッタがバッファ・エンプティ (送信できる)
1	0	1	$\overline{WAIT}$ (RxRDY)	L	レシーバにデータが準備できていない (受信できない)
				フローティング	レシーバにデータが準備できている (受信できる)
1	1	0	$\overline{RDY}$ (TxRDY)	L	トランスミッタがバッファ・エンプティ (送信できる)
				H	トランスミッタがバッファ・エンプティでない(送信できない)
1	1	1	$\overline{RDY}$ (RxRDY)	L	レシーバにデータが準備できている (受信できる)
				H	レシーバにデータが準備できていない (受信できない)

けなので，16バイト(8個のベクタ×2バイト・ベクタ)が繰り上がるようなアドレス(xxF1h～xxFFh)を開始アドレスにすることはできません．

このため当然のことながら，チャネルAとBでまったく別々のアドレスに割り込みベクタ・テーブルを設定することはできません．

● **書き込みレジスタ3(WR_3)**

WR_3はレシーバの機能を設定します．

▶ D_0：イネーブル・レシーバ

このビットが“1”であるとレシーバがイネーブルされ受信動作が可能になります．ただし，オート・イネーブル(WR_2のD_5)がセットされていると，$\overline{DCD}$入力端子がアクティブ(Lレベル)でないとデータは受信されません．このビットは受信に関するすべての設定が完了した後にセットします．

▶ D_1：SYNCキャラクタのロード禁止

このビットが“1”にセットされているとき，同期モードにおいてSYNCキャラクタ(WR_6，WR_7に設定されているデータ)，HDLCモードにおいてフラグ(WR_7に設定されているデータ)を受信しても受信FIFOにはロードされません．

つまり，これらのデータをデータ・レジスタに受信されないようにします．

この機能は同期またはHDLCモードで，受信データからSYNCキャラクタまたはフラグ・シーケンスを除外する手間を省くためのものです．

すが，$\overline{WAIT}$出力がLレベルの間はCPUを停止させられてしまうので，割り込みを受け付けなくなります．

これではCPUをデータの送信または受信のどちらかだけで使う場合を除いては，あまり使いものになりません．送信または受信のみで使うにしてもプログラムがIN命令またはOUT命令を繰り返していないと送受信できませんから，CPUは送受信以外の処理を行うことができなくなります．もしも，高速通信が必要な場合は次に述べるレディ機能を使いDMA転送で行うほうが得策です．

● **レディ機能(WR_1のD_6=“1”のとき)**

レディ機能を用いると，$\overline{W/RDY}$端子はDMAに対するレディ出力になります．

WR_1のD_5が“0”のときはレディ出力はトランスミッタに対して用いられ，送信バッファ・エンプティ(TxRDY)になるとLレベルになり，送信バッファ・エンプティでないとHレベルになります［図A(a)］．

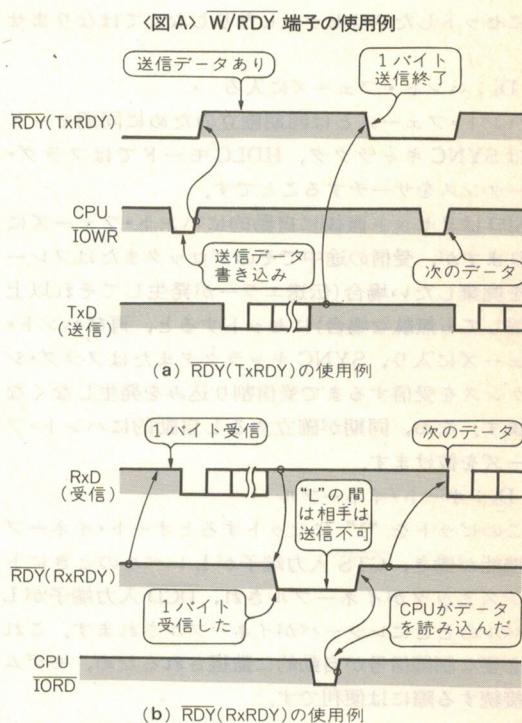
WR_1のD_5が“1”のときはレディ出力はレシーバに対して用いられ，受信データ有効(RxRDY)になるとLレベルになり，受信データ有効でないとHレベルになります．

これらのレディ出力をZ80DMAのRDY入力端子に接続して，LレベルでアクティブになるようにDMAをプログラムしておけば，TxRDYになったときに自動的にDMAによりメモリのバッファからデータが送信され，または，RxRDYになったときに自動的にDMAにより受信データがメモリのバッファに受信されるようになります．

送信と受信は同時にDMAによる転送はできませんが，例えば送信を割り込みで，受信をDMAで転送す

〈図A〉 $\overline{W/RDY}$ 端子の使用例

(a) $\overline{RDY}$(TxRDY)の使用例

(b) $\overline{RDY}$(RxRDY)の使用例

ることにより，CPUは送受信のためのSIOのポーリングから開放されます．

またレディ機能はDMA以外にも，シリアル入出力対応のワンチップ・マイコンなどとの通信のとき，1バイトごとのハンドシェイク信号としても使用できます．

▶ D_2：アドレス・サーチ・モード

このビットは，HDLCモードで従局(二次局)をプログラムする際に，WR_6に設定されたアドレスまたはグローバル・アドレス(FFh)をもつフレーム以外を無視したいときに"1"にセットします．

HDLCの従局では自分のアドレス以外のフレームを無視しなければならず，この機能は関係ないフレームでの受信割り込みまたは受信データの発生を阻止し，CPUが効率よく他の処理ができるように設けられたものです．

▶ D_3：レシーバCRCイネーブル

このビットがセットされていると，受信シフトレジスタから受信FIFOに最後の受信データ・ビットが転送されるときにCRCの計算を始めます．BSCなどの通信プロトコルではブロックの途中からCRC計算を始めなければならず，開始位置も透過モードと非透過モードで異なりますから，このビットはデータの受信中にセットしたりリセットしたりしなくてはなりません．

▶ D_4：ハント・フェーズに入る

ハント・フェーズとは同期確立のために同期モードではSYNCキャラクタ，HDLCモードではフラグ・シーケンスをサーチすることです．

SIOはリセット直後に自動的にハント・フェーズに入りますが，受信の途中でそのブロックまたはフレームを廃棄したい場合(伝送エラーが発生してそれ以上受信しても無駄な場合)にセットすると，再びハント・フェーズに入り，SYNCキャラクタまたはフラグ・シーケンスを受信するまで受信割り込みを発生しなくなります．なお，同期が確立すると自動的にハント・フェーズを抜けます．

▶ D_5：オート・イネーブル

このビットを"1"にセットするとオート・イネーブル機能が働き，$\overline{CTS}$入力端子がLレベルのときにトランスミッタがイネーブルされ，$\overline{DCD}$入力端子がLレベルのときにレシーバがイネーブルされます．これは必要な制御信号が自動的に監視されるため，モデムを接続する際には便利です．

このビットがセットされていると，CSがオフのときは送信が自動的に止まり，CDがオフのとき(つまりキャリアが届いていないとき)の受信データはノイズによるものなので自動的に無視するようになります．ハードウェア・フロー制御(RS-CSフロー)のときに便利です．

▶ D_6～D_7：受信キャラクタ長

これら2ビットにより受信キャラクタのビット長を設定します．なお，受信キャラクタ長が7ビット以下の場合はデータは右づめ(下位ビット側)でデータ・レジスタにロードされ，その左にパリティ・ビットが付加されるので，もとのデータを再現するためにはパリティ・ビットをマスクしなければなりません．

● **書き込みレジスタ4(WR_4)**

このレジスタはトランスミッタとレシーバの両方に共通なパラメータを設定します．

▶ D_0：パリティ・イネーブル

このビットを"1"にセットすると，トランスミッタのパリティ・ジェネレータおよびレシーバのパリティ・チェッカがイネーブルされ，送信データにパリティ・ビットが付加され，受信データのパリティ・ビットがチェックされます．

▶ D_1：偶数/奇数パリティの選択

このビットを"1"に設定すると偶数，"0"に設定すると奇数パリティとなります．なお，パリティ・イネーブルがディセーブルのとき(WR_4のD_0が"0")はこのビットは"1"でも"0"でも無効です．

▶ D_2～D_3：ストップ・ビット長

これらのビットにより非同期モードでのストップ・ビット長を設定します．また，同期モードおよびHDLCモードの場合は，両方とも"0"に設定する必要があります．ストップ・ビット長は大は小を兼ねる特徴があり，送信データのストップ・ビットが長くても単にキャラクタの間隔とみなされ(伝送効率は悪くなる)，受信データのストップ・ビットが足りなくてもパリティを使わない場合はエラーは発生しません．見落としやすい点ですから注意しましょう．

▶ D_4～D_5：同期モード

これらのビットは同期モード(WR_4のD_2="0"，D_3="0")のときの同期モードの種類を設定します．

▶ D_6～D_7：送受信クロックの倍率

これらのビットは送受信クロック(TxC，RxC入力端子)のデータ送受信レートに対する倍率を設定します．同期モードおよびHDLCモードでは1倍に設定しなければなりません．また非同期モードでは原則的に16倍以上を設定しなければなりません．

● **書き込みレジスタ5(WR_5)**

WR_5はトランスミッタのパラメータを設定します．

▶ D_0：トランスミッタCRCイネーブル

このビットが"1"にセットされていると，トランスミッタのCRCジェネレータがイネーブルされます．送信データが送信FIFOから送信シフトレジスタにロードされたときにこのビットが"1"であるとそのデータに対してCRCが計算されます．また，このビットが"1"にセットされていて，同期モードまたはHDLCモードのときに送信アンダーラン(送信すべきデータがない)が発生すると，自動的にCRCが送信されます．

▶ D_1：RTS端子制御

このビットが"1"にセットされていると$\overline{RTS}$端

子にLレベルが，“0”にセットされるとHレベルが出力されます．ただし非同期モードのとき，“0”にセットしていても，送信バッファに送信すべきデータがあるとLレベルになり，送信バッファが空になってからHレベルになります．

これは$\overline{RTS}$端子の本来の動作で，送信すべきデータがあるときLレベルになります．現在の全二重通信では，この端子をRS-CSフロー制御などに使い，受信バッファがオーバしそうなときHレベルにして相手の送信を止める意味でも使われています．

▶ D_2：CRC多項式の選択

このビットが“0”に設定されているとCRCはCCITT$(X^{16}+X^{12}+X^5+1)$が，“1”に設定されているとCRC16$(X^{16}+X^{15}+X^2+1)$が選択されます．BSCではCRC16が，HDLCではCCITT方式が一般的です．

▶ D_3：トランスミッタ・イネーブル

このビットが“1”にセットされているとトランスミッタがイネーブルされます．データの送信中に“0”に設定しても送信が終わるまではイネーブルのままです．

オート・イネーブルがセットされている状態で実際に送信するためには，$\overline{CTS}$入力端子がLレベルでなければなりません．

▶ D_4：ブレーク信号の送信

このビットを“1”にセットすると，どのようなデータを送信していても送信データ(TxD)は強制的にスペース(“0”＝Lレベル)にされます．

一般的にブレーク信号は相手の送信を強制的に停止させたい場合に用いられます．実際にブレーク信号として機能させるためには，キャラクタ長＋スタート・ビット＋ストップ・ビット長よりも長くスペース状態を維持する必要があります．

▶ D_5～D_6：送信キャラクタ長

これらのビットは送信データのビット長を設定します．5ビット以下の設定の場合は，実際のデータが何ビットであるかをSIOが判別できるように**表4**のように実際のビットの左側に“000”を付加し，さらにその左側に(5－実際のビット数)の“1”を付加します．

▶ D_7：DTR端子制御

$\overline{DTR}$端子の状態を設定します．“1”を書き込むとLレベルが，“0”を書き込むとHレベルが出力されます．この端子は非同期モードでも，送信バッファの状態に関係なくいつでも自由に設定できます．

● 書き込みレジスタ6(WR_6)

WR_6には同期モードではSYNCキャラクタ(バイ

SIOの送受信クロック倍率

SIOの送受信クロック倍率は，16倍/32倍/64倍の3通りの設定が可能ですが，実際にはどれを使ったら良いのか迷ってしまいます．とくにCTCと組み合わせて使う場合にはCTCの設定によってもソフトウェア的に変更できるのでなおさらです．

SIOの送受信クロック倍率は，理論的には大きいほうが受信マージン(受信データ・ビットのひずみに対する余裕度)が大きくなって良いのですが，実際にはどちらでもほとんど違いはありません．

そこで，最高データ・レートと最小データ・レートを考慮して決定するのが基本になります．

現在使われるであろう最低は300 bpsで，最高は速ければ速いほど良いと思われます．RS-232-C規格の最高は20 kbpsですが，最近はこれよりも高速なモデムもありますから，やはり最高は速ければ速いほど良いでしょう．

参考までに，CPUクロックと使うことができる通信速度を表にまとめておきます．表中でアミがかかっている部分は，周波数が正確に合っているわけではないので，同期モードでは使用できません．

〈表B〉通信速度とCTCの設定

通信速度	CPUクロック					
	4 MHz	4.9152 MHz	6 MHz	8 MHz	9.8304 MHz	10 MHz
38400 bps	×	×	×	×	1	×
28800 bps	×	×	×	×	×	×
19200 bps	×	1	×	×	2	×
14400 bps	×	×	×	×	×	×
9600 bps	×	2	×	×	4	×
4800 bps	×	4	×	×	8	×
2400 bps	×	8	×	13	16	16
1200 bps	13	16	×	26	32	33
600 bps	26	32	39	52	64	65
300 bps	52	64	78	104	128	130
200 bps*	78	96	117	156	192	195
150 bps*	104	128	156	208	256(0)	△
110 bps*	142	174	213	△	△	△
75 bps*	208	256(0)	△	△	△	△
50 bps*	×	△	△	△	△	△

*ほとんど使われない
数字：CTCタイマ・モード，プリスケーラ16でSIOクロック倍率16のときのCTC分周比
( )内はレジスタに書き込む値
△：SIOクロック32倍クロック設定で可能
×CTCのタイマ・モード(プリスケーラ16または256分周)では不可

シンクの場合は1バイト目)を設定します．設定されたデータはSYNCキャラクタのロード禁止(WR_3のD_1が“1”)の場合は受信データにはなりません．しかし，送信時にはプログラムで送信データ・レジスタに書き込まなければなりません．

HDLCモードでアドレス・サーチ・モード(WR_3のD_2が“1”)の場合，従局アドレスを設定しておくと，そのアドレスまたはグローバル・アドレス(FFh)以外のアドレスをもつフレーム以外は受信データになりません．非同期モードでは設定する必要はありません．

● **書き込みレジスタ7(WR_7)**

WR_7には同期モードでバイシンク(16ビットSYNCキャラクタ)の場合に2バイト目のSYNCキャラクタを設定します．しかし，送信時にはプログラムで送信データ・レジスタに書き込まなければなりません．HDLCモードではフラグ・シーケンスのパターン(7Eh)を設定します．

HDLCモードではトランスミッタ・イネーブルのときに送信するデータがなくなると，(送信CRCイネーブルのときはCRCに続いて)自動的にフラグ・シーケンスが送信されます．

読み出しレジスタの詳細

● **読み出しレジスタ0(RR_0)**

RR_0はSIOの基本的な状態を表します．

▶ D_0：受信キャラクタ有効

このビットは受信FIFOに1キャラクタ以上の受信データがあるときに“1”になります．CPUが1キャラクタを読み込んでも，受信FIFOにまだデータがある場合には“0”になりません．

▶ D_1：割り込み保留中

このビットはSIOが割り込みを発生する何らかの状態をもっているときに，割り込みの種類に関係なく“1”になります．なお，このビットはチャネルAのRR_0にしかありませんので，チャネルBの割り込み保留中を確認する場合でもチャネルAのRR_0を読み込まなければなりません．

▶ D_2：送信バッファ・エンプティ

このビットは送信バッファ(FIFO)に空きができると“1”になります．SIOはリセットされるとこのビットが“1”になります．しかし，送信データを書き込んでもSIOの初期化がされていないと(オート・イネーブルで$\overline{CTS}$端子がHレベルのときも)永久に“0”になったままです．

▶ D_3：DCD(データ・キャリア検出)の状態

このビットは最初に変化した後の$\overline{DCD}$入力端子の状態を表します．$\overline{DCD}$端子がHレベルからLレベルに変化すると，このビットは“1”になり，Lレベルからで Hレベルに変化すると“0”になります．以後，外部/ステータス割り込みのリセット・コマンド(WR_0に10hを書き込む)が実行されるまで，この状態はラッチされ変化しません．したがって，最新の状態を読み込みたいときは，外部/ステータス割り込みのリセット・コマンドを実行する必要があります．このラッチ動作は以下のD_4からD_7にも適用されます．

〈表4〉送信データが5ビット以下の場合

キャラクタ当たりの送信ビット数(キャラクタ長)	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	1	1	1	0	0	0	D_0
2	1	1	1	0	0	0	D_1	D_0
3	1	1	0	0	0	D_2	D_1	D_0
4	1	0	0	0	D_3	D_2	D_1	D_0
5	0	0	0	D_4	D_3	D_2	D_1	D_0

D_n：データ・ビット

▶ D_4：シンク/ハント

このビットはモードにより意味が異なります．

非同期モードでは$\overline{SYNC}$端子の状態を表します．

内部同期モードではエンター・ハント・フェーズ(WR_3のD_4をセット)によりセットされ，SIOが同期キャラクタ(モノシンクのときはWR_6，バイシンクのときはWR_6とWR_7が連続して)が受信されたときにリセットされて“0”になります．

HDLCモードではエンター・ハント・フェーズによりセットされ，SIOがオープニング・フラグ(WR_7のデータ=7Eh)が受信されたときにリセットされて“0”になります．

外部同期モードでは外部に設けられた同期検出回路の出力を$\overline{SYNC}$端子に入力しますが，その$\overline{SYNC}$端子が変化したときに変化後の状態を表します．

▶ D_5：CTS(クリア・ツー・センド)の状態

このビットは最初に変化した後の$\overline{CTS}$入力端子の状態を表します．$\overline{CTS}$端子がHレベルからLレベルに変化すると，このビットは“1”になります．LレベルからHレベルに変化すると“0”になります．しかし，変化するのは最初の1回のみで，最新の状態を調べたい場合は外部/ステータス割り込みのリセット・コマンドを実行する必要があります．

▶ D_6：送信アンダーラン/EOM

SIOがリセットされると，このビットは“1”にセットされます．このビットをリセットする唯一の方法はリセット・トランスミッタ・アンダーラン・コマンドを実行する(WR_0にC0hを書き込む)ことです．また，送信FIFOが空になっても送信データを書き込まないと，送信アンダーランが発生してこのビットがセットされ，同時に外部/ステータス割り込みが発生します．

同期モードではこのビットがセットされていると，送信アンダーランが発生しても外部/ステータス割り込みを発生せず，アイドル・シンク・キャラクタ(WR_6のデータ)を送信して同期確立を維持します．同期モードでこのビットがリセットされていると，送信FIFOにデータがなくなると2バイトのCRCを送信し，外部/ステータス割り込みを発生します．BSCなどでCRCを送信したいときはデータ・ブロックの最後(ETX送信後など)でこのビットをリセットしておけば自動的にCRCが送信されます．

読み出しレジスタのCPUへの読み込みはIN命令を使います．

HDLCモードではこのビットがセットされていると，送信アンダーランが発生しても外部/ステータス割り込みを発生せずに，フラグ・シーケンス(WR_7のデータ＝7Eh)を送信し続けて同期を維持します．しかし，HDLCではフレームの途中にフラグ・シーケンスが挿入されると，それをクロージング・フラグと解釈して無効フレーム(ビット数が少ないかCRCに誤りがある)と判断され廃棄されますので，送信アンダーランを起こさないように送信しなければなりません．どうしても送信を中断しなくてはならない場合にはアボート・シーケンスを送信しなくてはなりません．

HDLCモードでこのビットがリセットされていると，送信アンダーランが発生した時点でフレームの終わりと見なされてCRC(FCS)が自動的に送信され，同時に外部/ステータス割り込みが発生します．CRCの送信が終了すると送信バッファ・エンプティ割り込みが発生しますが，そこで保留中の送信割り込みリセット・コマンド(28h)をWR_0に書き込むと，それ以降は送信バッファ・エンプティ割り込みは発生せず，送信データを書き込まずにいればクロージング・フラグが送信され続けます．

HDLC回線を不活性化(フラグ・シーケンスを常に送らないで“1”に固定する)にはトランスミッタ・イネーブル(WR_5のD_3)を“0”にします．

▶ D_7：ブレーク/アボート検出

非同期モードでブレーク信号を受信するとセットされ，外部/ステータス割り込みを発生します．また，外部/ステータス割り込みをリセットしておくと，ブレーク信号が解除されたときにも外部/ステータス割り込みを発生します．

HDLCモードでアボート・シーケンスを受信するとセットされ，外部/ステータス割り込みを発生します．また，外部/ステータス割り込みをリセットしておくと，アボート状態が解除されたときにリセットされ，外部/ステータス割り込みを発生します．

● **読み出しレジスタ1(RR_1)**

このレジスタには特殊受信状態割り込みの原因などが設定されます．

▶ D_0：全部送信し終わった

非同期モードにおいて送信FIFOの全データを送信し終わると“1”になります．同期モードおよびHDLCモードでは常に“1”になります．半二重通信で送信から受信に切り替える(RSをオフにする)前に，このビットが“1”になるのを確認します．

▶ D_1～D_3：レシデュー・コード

HDLCでは伝送するデータはキャラクタ単位ではなくビット単位に可能で，たまたま8ビットごとに伝送しているだけですから，Iフレームを受信した場合の最後の8ビットはすべて有効なビットとは限りません．

そこで，最後のデータとその直前のデータの有効ビット数をこれらのビットで表します．これらのビットが意味をもつのはエンド・オブ・フレーム(RR_1のD_7が“1”)のときだけです．

▶ D_4：パリティ・エラー

このビットはパリティがイネーブルされているときに受信データ中にパリティ・エラーが検出されると“1”にセットされます．いったんセットされるとエラー・リセット・コマンドが実行(WR_0に30hを書き込む)されるまでそのままです．したがって，データ・ブロックの受信が終わってからエラー・チェックをすることが可能です．

▶ D_5：オーバーラン・エラー

受信FIFOがいっぱいのときに，さらに次のデータが受信されると“1”になります．ただし実際にエラーがセットされるのは，実際にオーバ・ランが発生したデータが読み込まれたときにセットされます．このビットがセットされたときは受信データのCPUへの取り込みが間に合わず受信データを失った，つまり受信データは相手が送ったデータより少ないことを意味します．

いったんセットされるとエラー・リセット・コマンドが実行されるまでそのままです．

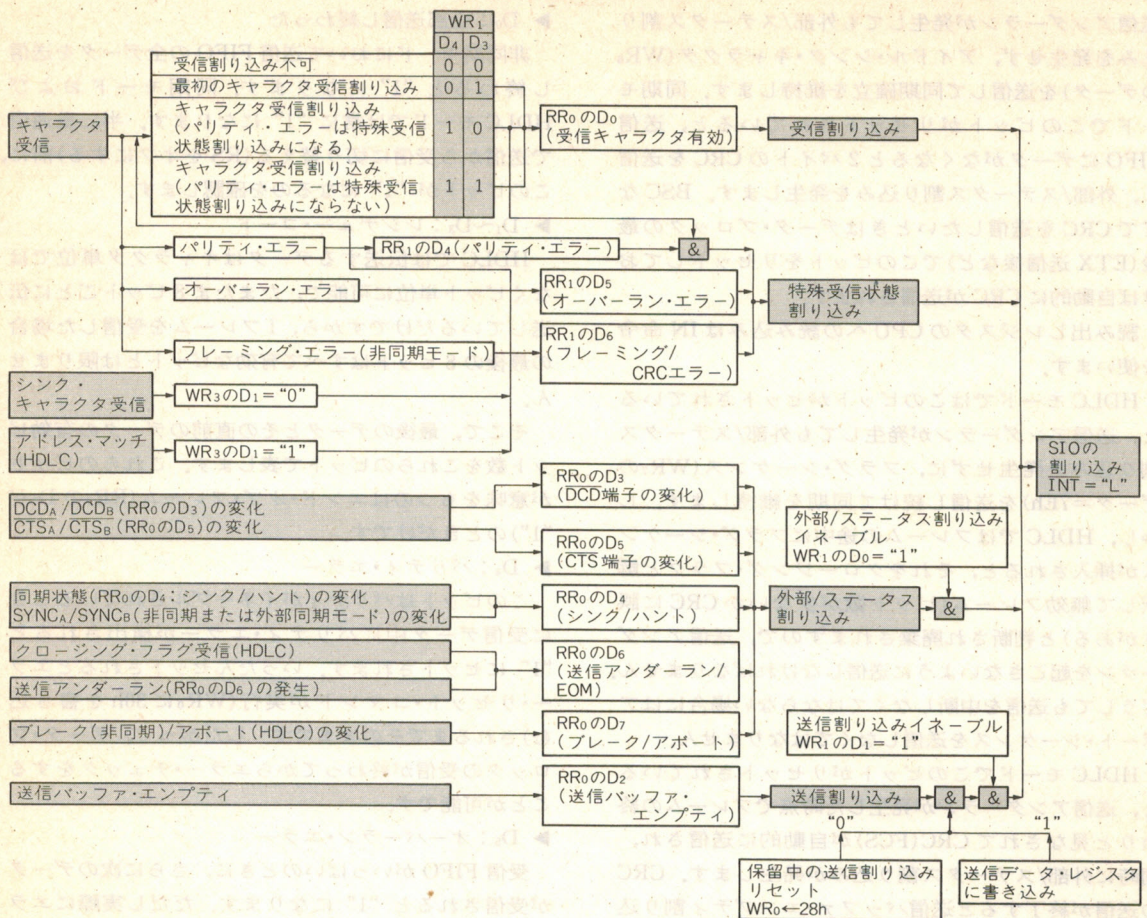
▶ D_6：CRC/フレーミング・エラー

非同期モードでは受信データにフレーミング・エラー(ストップ・ビットであるべきビットがデータであった)が発生したことを，同期モードおよびHDLCモードではそれまでのCRCチェックに誤りがあったことを表します．

いったんセットされるとエラー・リセット・コマンドが実行されるまでそのままです．

▶ D_7：エンド・オブ・フレーム(EOM)

このビットはHDLCモードのみで使用されます．これはフレームの受信が正常に(CRCエラーなしでクロージング・フラグを受信)終了したときにセットされ，次のフレームの最初のデータでリセットされます．

<図6> Z80SIO の割り込みシステム

● 読み出しレジスタ 2(RR_2)

このレジスタはチャネルBにしかありません．このレジスタはCPUに対して発生した割り込みのベクタを保持します．これはCPUがZ80以外の場合や割り込みモードが2以外の場合に，バスにベクタは送れないが，割り込み原因の解析を簡単に行いたい場合に読むものです．よほどの理由がない限り，ステータス・アフェクツ・ベクタ機能により割り込みベクタを変えて要因を判定するのが一般的で，このレジスタを読み込むことはないでしょう．

SIOのデータ送受信割り込み

Z80SIOは他のCTCやPIOに比較して非常に高機能で，いろいろな割り込みを発生させることができます．ここでZ80SIOの割り込みについて図6にまとめてみます．

割り込みには大きく分けて，受信割り込み，送信割り込み，外部/ステータス割り込み，特殊受信状態割り込みの四つがあります．

またこれらの四つの割り込みは，ステータス・アフェクツ・ベクタ(WR_1のD_2)がセットされていれば，チャネルごとにそれぞれ独立した割り込みベクタを発生するので，割り込みルーチンの先頭での割り込み要因判別ルーチンが不要になります．

● 受信割り込みのいろいろ

図7をみてください．一般に非同期のシリアル通信では，キャラクタとキャラクタの間隔は任意です．いつ受信されるかわからないデータをポーリングで待つのは不利です．受信割り込みとはこのように，非同期に送られてくるデータの受信に最適で，データが送られてくる間はCPUは別のことを処理できます．

SIOの受信割り込みは3通り設定できます．一般的にはキャラクタ受信時の割り込み(パリティ・エラーは特殊受信状態になる)を使用します．

Z80SIOではパリティ・エラーの扱いが少し特殊で，キャラクタ受信割り込みに入れることも，特殊受信状態割り込みに入れることもできます．当然キャラクタ

受信割り込みに入れると，パリティ・エラーで特殊受状態信割り込みは発生しません．ただしパリティ・エラーのビット(RR_1のD_4)は，どちらに設定されていても，パリティ・エラーが発生すると“1”になります．

● 最初のキャラクタのみ割り込み

“最初のキャラクタの割り込み”と聞いても何のことかよくわからない方もいるでしょう．図7(b)のように，データがキャラクタ単位ではなく，あるブロックごとに固まって送られてくる場合があります．このときは最初のキャラクタのみ，割り込みで受信し，後のデータはポーリングで受信するという方法もあります．

しかし最初のキャラクタ受信割り込みで次のブロック・データをポーリングする(RR_0のD_0が“1”になるのを待つ)方法では，その間に別の割り込み(タイマ割り込みやSIOの他チャネルの割り込みなど)の処理ができません．筆者の考えとしてはあまり有効な受信割り込みとは思えません．

普通にキャラクタ受信割り込みを使うのが一般的だと思います．

● 送信割り込み

受信割り込みは，相手が送ってきたデータを受信したときに発生する割り込みですが，送信割り込みとは，自分の送信バッファに空きができたとき発生する割り込みです．つまり，例えば100バイト送信したら終わりというときでも，100バイト目を送信し終えた後で割り込みが発生します．もう送信するデータがないのにもかかわらずです．

保留中の送信割り込みリセット(WR_0に28hを書き込む)とはこのように，送信割り込みが発生したにもかかわらず，送信すべきデータがなくなったときに行います．

このコマンドを実行せずに，しかも送信データを何も書き込むことなく割り込みルーチンから復帰(RETI命令を実行)すると，すぐにまた送信割り込みが発生し，結果的にメイン・ルーチンが動けなくなります．

次に送信したいデータが発生した場合には，そのまま送信したいデータをデータ・ポートに書き込むだけでOKです．そのデータが送信されて，送信バッファがエンプティになると，また再び送信割り込みを発生するようになります．

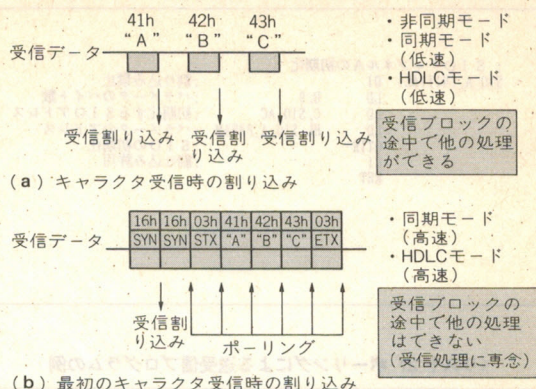
● 特殊受信状態/外部/ステータス割り込み

受信エラーのときは特殊受信状態割り込みが発生しますが，何のエラーかまで判別したいときは，RR_1を読み出して判別します．

同じように，外部/ステータス割り込みについても，RR_0を読み出せば詳しい割り込み発生の原因がわかります．

〈図7〉キャラクタ受信割り込みと最初のキャラクタ受信割り込み

(a) キャラクタ受信時の割り込み

(b) 最初のキャラクタ受信時の割り込み

またこれらの割り込みはWR_0に外部/ステータス割り込みリセット，もしくはエラー・リセット・コマンドを書き込まないと，再度割り込みが発生しないので注意してください．

SIOのプログラミング

● SIOの初期化

SIOのレジスタ初期化には，リスト1のようにOTIR命令(Bレジスタが0になるまで，HLレジスタが示すアドレスのデータをCレジスタが示すポートに出力し，HLレジスタを+1する)を使うのが便利です．

リストの最初のDI命令は，初期化中に割り込みが発生しないようにするための安全策です．

実際のパラメータは，リスト1のようにレジスタ・ポインタ・コマンドとそのレジスタへのパラメータを順番に並べます．

● 3線式シリアル伝送

SIOをモデムに接続するのではなく単純なシリアル伝送(送受信)に使用するならば，線が3本(TxD, RxD, GND)の3線式シリアル伝送方式で十分です．

この場合，SIOの$\overline{CTS}/\overline{DCD}$端子はオート・イネーブルが設定されても動作するように，GNDに接続しておくのが安全です．

どうしても入力端子を汎用の入力ポートとして使いたいときは，オート・イネーブルに設定してはなりません．

図8に回路例を，リスト2にプログラム例を示します．

● ポーリングによるデータ送受信

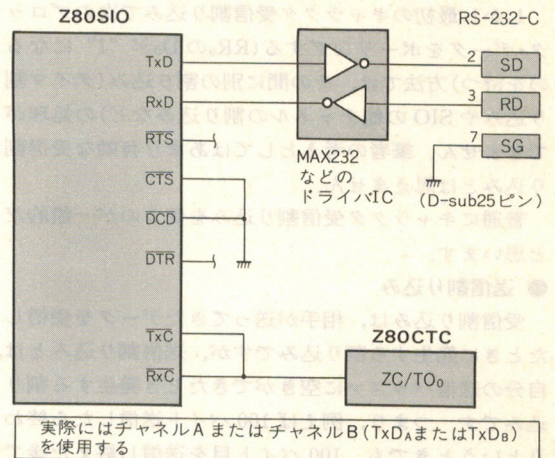
データの送信をするにはSIOを初期化しイネーブル・トランスミッタをセットします．そしてまず最初にRR_0のビット2(送信バッファ・エンプティ)を調べて“0”ならば“1”になるまで待ちます．

〈リスト1〉SIOのチャネル初期化プログラム例

```
; SIOチャンネルAの初期化
SIO_A_INITIAL:  DI                      ;割り込み禁止
                LD      B,9             ;パラメータのバイト数
                LD      C,SIO_AC        ;初期化するSIOアドレス
                LD      HL,SIO_A_PARAM  ;パラメータのアドレス
                OTIR                    ;SIOの初期化
                EI                      ;割り込み許可
                RET

; SIOのパラメータ
SIO_A_PARAM:    DB      18H             ;チャネル・リセット
                DB      1               ;WR1 送信割り込みなし
                DB      18H             ;キャラクタ受信時の割り込み
                DB      4               ;WR4  CLOCK *16
                DB      47H             ;STOP bit 1  偶数パリティ
                DB      3               ;WR3  受信キャラクタ 7bit
                DB      41H             ;イネーブル・レシーバ
                DB      5               ;WR5  送信キャラクタ 7bit
                DB      0AAH            ;イネーブル・トランスミッタ
```

〈リスト2〉ポーリングによる送受信プログラムの例

```
; SIOポーリング送受信プログラム (3線式)

;       I/Oアドレス定義
SIO_AD          EQU     18H             ;SIOチャネルAデータ
SIO_AC          EQU     19H             ;SIOチャネルAコントロール
SIO_BD          EQU     1AH             ;SIOチャネルBデータ
SIO_BC          EQU     1BH             ;SIOチャネルBコントロール
CTC_C0          EQU     10H             ;CTCチャネル0

                ORG     0               ;プログラム実行開始アドレス
                LD      SP,0000H        ;スタック・ポインタの設定
                CALL    CTC_0_INITIAL   ;CTCの初期化
                CALL    SIO_A_INITIAL   ;SIOの初期化
                JP      MAIN            ;メイン・ルーチンへジャンプ
; CTCを使った送受信クロックの設定 (例)
CTC_0_INITIAL:  LD      A,47H           ;CTCのモード・コマンド
                OUT     (CTC_C0),A      ;モード・コマンドを出力
                LD      A,16            ;分周比
                OUT     (CTC_C0),A      ;分周比を出力 (設定)
                RET
; SIOチャネルAの初期化
SIO_A_INITIAL:  DI                      ;割り込み禁止
                LD      B,7             ;パラメータのバイト数
                LD      C,SIO_AC        ;初期化するSIOアドレス
                LD      HL,SIO_A_PARAM  ;パラメータのアドレス
                OTIR                    ;SIOの初期化
                EI                      ;割り込み許可
                RET
; SIOのパラメータ
SIO_A_PARAM:    DB      18H             ;チャネル・リセット
                DB      4               ;WR4  CLOCK *16
                DB      47H             ;STOP bit 1  偶数パリティ
                DB      3               ;WR3  受信キャラクタ 7bit
                DB      41H             ;イネーブル・レシーバ
                DB      5               ;WR5  送信キャラクタ 7bit
                DB      0AAH            ;イネーブル・トランスミッタ

; 1キャラクタ送信 (ポーリング)
SEND_BYTE:      PUSH    AF              ;Aレジスタ/フラグの保存
SEND_BYTE_1:    IN      A,(SIO_AC)      ;RR0をAレジスタに読み込み
                AND     4               ;送信バッファ・エンプティ
                JR      Z,SEND_BYTE_1   ; "0" の場合は待つ
                POP     AF              ;Aレジスタ/フラグを復元
                OUT     (SIO_AD),A      ;送信データを書き込み
                RET                     ;サブルーチンからの復帰

; 1キャラクタ受信 (ポーリング)
RECEIVE_BYTE:   IN      A,(SIO_AC)      ;RR0をAレジスタに読み込み
                AND     1               ;受信データ有効
                JR      Z,RECEIVE_BYTE  ; "0" の場合は待つ
                IN      A,(SIO_AD)      ;受信データを読み込み
                RET                     ;サブルーチンからの復帰

; RR1をAレジスタに読み込み
INPUT_RR1:      DI                      ;割り込み禁止
                LD      A,1             ;ポインタ・コマンド
                OUT     (SIO_AC),A      ;コマンド実行 (RR1選択)
                IN      A,(SIO_AC)      ;エラー情報 (RR1) の読み込み
                EI                      ;割り込み許可
                RET

; コマンド (WR0) 実行の例 (エラー・リセット)
RESET_ERROR:    LD      A,30H           ;エラー・リセット・コマンド
                OUT     (SIO_AC),A      ;コマンド実行
                RET                     ;サブルーチンからの復帰
```

〈図8〉3線式のRS-232-Cインターフェース回路例

送信バッファ・エンプティならばSIOのデータ・ポートに送信したいデータを書き込むと送信されます．ただしオート・イネーブルがセットされているときは，$\overline{\text{CTS}}$端子がLレベルにならないと送信されません．

データの受信をするにはSIOを初期化しイネーブル・レシーバをセットしてから，RR_0のビット0(受信データ有効)を調べて，“0”ならば“1”になるのを待ちます．

受信データ有効ならばSIOのデータ・ポートから受信データを読み込みます．またオート・イネーブルがセットされているときは，$\overline{\text{DCD}}$端子がLレベルになっていないとデータ受信ができません．

図9に，いちばん基本的なSIOの送受信設定のチェック・ポイントを示します．少なくとも非同期通信3線式ポーリング送受信であれば，これらの項目をきちんと設定できれば，データの送受信ができるはずです．

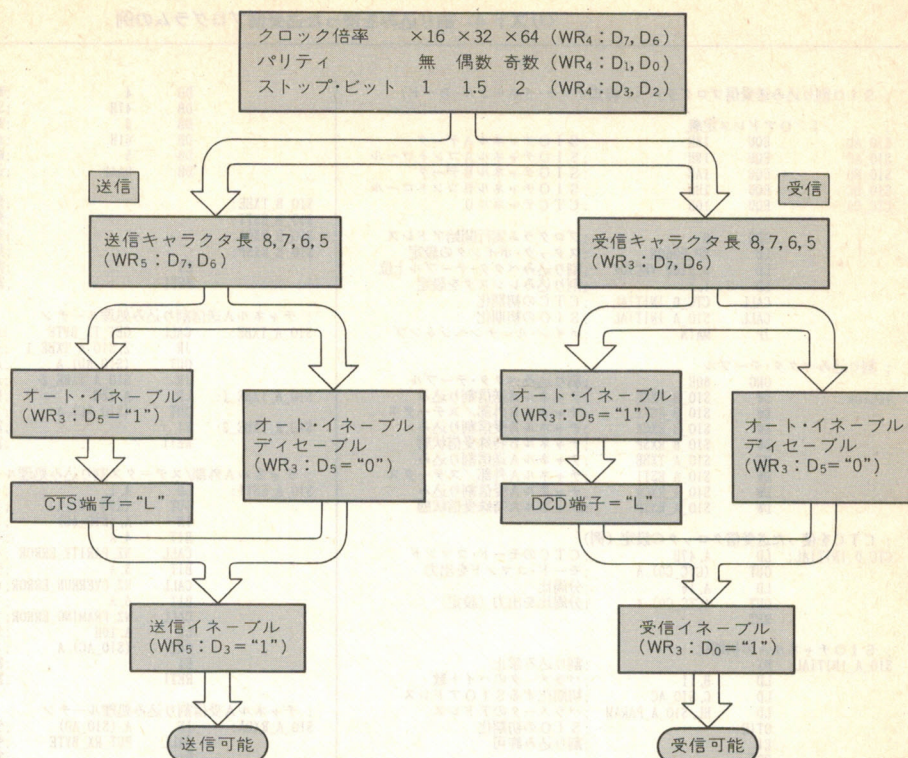
● チャネルAのDTRを制御する

リスト3に例を示します．WR_0以外のレジスタの設定を変える場合は，割り込みルーチン処理により別のレジスタに書き込まれるのを防ぐためにまず割り込みを禁止します．

〈図 9〉
SIO の基本設定
チェック・ポイント

クロック倍率　×16　×32　×64　(WR₄：D₇, D₆)
パリティ　無　偶数　奇数　(WR₄：D₁, D₀)
ストップ・ビット　1　1.5　2　(WR₄：D₃, D₂)

送信
送信キャラクタ長 8,7,6,5 (WR_5：D_7, D_6)
オート・イネーブル (WR_3：D_5＝"1")
$\overline{CTS}$端子＝"L"
オート・イネーブル ディセーブル (WR_3：D_5＝"0")
送信イネーブル (WR_5：D_3＝"1")
送信可能

受信
受信キャラクタ長 8,7,6,5 (WR_3：D_7, D_6)
オート・イネーブル (WR_3：D_5＝"1")
$\overline{DCD}$端子＝"L"
オート・イネーブル ディセーブル (WR_3：D_5＝"0")
受信イネーブル (WR_3：D_0＝"1")
受信可能

そしてコントロール・ポートに書き込みたいレジスタ・ポインタを出力することにより書き込みレジスタを選択し，次に設定したいデータを同じくコントロール・ポートに出力します．

またすでに述べたように，SIO は書き込みレジスタの現在の状態を読み出すことができません．他のビットの状態がわからず，またその状態を変えずに特定のビットだけを ON/OFF したいときは，メモリ上に書き込んだデータを保存しておき，そのデータに対して OR 命令や AND 命令で特定のビットをセットしたり，リセットしたデータを書き込みます．

もちろん，初期化した後なにも変更していないときなど，この例でいえば $\overline{DTR}$ 以外のビットの状態がわかりきっているときなどは，ワークエリアに保存などしないで，直接書き込んでもよいでしょう．

● 割り込みを使ったデータ送受信

リスト 4 に割り込みを使ったデータ送受信プログラムを，図 10 に回路例を示します．割り込みを使う場合は設定が必要なレジスタの数も増えます．

送信や受信は，割り込み発生により直接それぞれのルーチンにジャンプするので，ポーリングのときのような RR_0などのビットの状態をチェックする必要がなく，ポーリングのときより簡単になります．

● 割り込みと HDLC モードでの初期化

リスト 5 に割り込みを使った HDLC モードでのプ

〈リスト 3〉 $\overline{DTR}$ 端子をコントロールするプログラム例

```
; チャネルAのDTRをLレベル（ER信号をオン）
CH_A_ER_ON:     DI                      ;割り込み禁止
                PUSH    AF              ;Aレジスタ/フラグを保存
                LD      A,5             ;レジスタ・ポインタ・コマンド
                OUT     (SIO_AC),A      ;WR5を指定
                LD      A,(SIOA_WR5)    ;WR5保存データ読み出し
                OR      80H             ;bit7 ON (DTR = "L" )
                OUT     (SIO_AC),A      ;WR5を書き換え
                LD      (SIOA_WR5),A    ;WR5保存
                POP     AF              ;Aレジスタ/フラグを復元
                EI                      ;割り込み許可
                RET                     ;サブルーチンからの復帰

; チャネルAのDTRをHレベル（ER信号をオフ）
CH_A_ER_OFF:    DI                      ;割り込み禁止
                PUSH    AF              ;Aレジスタ/フラグを保存
                LD      A,5             ;レジスタ・ポインタ・コマンド
                OUT     (SIO_AC),A      ;WR5を指定
                LD      A,(SIOA_WR5)    ;WR5保存データ読み出し
                AND     07FH            ;bit7 OFF (DTR = "H" )
                OUT     (SIO_AC),A      ;WR5を書き換え
                LD      (SIOA_WR5),A    ;WR5保存
                POP     AF              ;Aレジスタ/フラグを復元
                EI                      ;割り込み許可
                RET                     ;サブルーチンからの復帰
```

ログラムを，図 11 にインターフェースの回路例を示します．

ポーリングのときとの違いは，WR_2に割り込みベクタ・テーブルの下位 8 ビットを設定することと，WR_1で割り込みをイネーブルしていることです．

実際の割り込み処理ルーチンはポーリングのときと同じでかまいません．この初期化例でとくに注意することは，チャネル A しか使っていないにもかかわら

〈リスト4〉割り込みを使った送受信プログラムの例

```
; SIO割り込み送受信プログラム（7線式オート・イネーブル・モード）

;          I／Oアドレス定義
SIO_AD         EQU     18H              ;SIOチャネルAデータ
SIO_AC         EQU     19H              ;SIOチャネルAコントロール
SIO_BD         EQU     1AH              ;SIOチャネルBデータ
SIO_BC         EQU     1BH              ;SIOチャネルBコントロール
CTC_C0         EQU     10H              ;CTCチャネル0

               ORG     0                ;プログラム実行開始アドレス
               LD      SP,0000H         ;スタック・ポインタの設定
               LD      A,HIGH VECTOR    ;割り込みベクタ・テーブル上位
               LD      I,A              ;割り込みレジスタを設定
               CALL    CTC_0_INITIAL    ;CTCの初期化
               CALL    SIO_A_INITIAL    ;SIOの初期化
               JP      MAIN             ;メイン・ルーチンへジャンプ

; 割り込みベクタ・テーブル
               ORG     80H              ;割り込みベクタ・テーブル
VECTOR:        DW      SIO_B_TXBE       ;チャネルB送信割り込み
               DW      SIO_B_ESTI       ;チャネルB外部／ステータス
               DW      SIO_B_RXCA       ;チャネルB受信割り込み
               DW      SIO_B_RXSP       ;チャネルB特殊受信状態
               DW      SIO_A_TXBE       ;チャネルA送信割り込み
               DW      SIO_A_ESTI       ;チャネルA外部／ステータス
               DW      SIO_A_RXCA       ;チャネルA受信割り込み
               DW      SIO_A_RXSP       ;チャネルA特殊受信状態

; CTCを使った送受信クロックの設定（例）
CTC_0_INITIAL: LD      A,47H            ;CTCのモード・コマンド
               OUT     (CTC_C0),A       ;モード・コマンドを出力
               LD      A,16             ;分周比
               OUT     (CTC_C0),A       ;分周比を出力（設定）
               RET

; SIOチャネルAの初期化
SIO_A_INITIAL: DI                       ;割り込み禁止
               LD      B,11             ;パラメータのバイト数
               LD      C,SIO_AC         ;初期化するSIOアドレス
               LD      HL,SIO_A_PARAM   ;パラメータのアドレス
               OTIR                     ;SIOの初期化
               EI                       ;割り込み許可
               RET

; SIOのパラメータ
SIO_A_PARAM:   DB      18H              ;チャネル・リセット
               DB      2                ;WR2
               DB      LOW VECTOR       ;割り込みベクタ設定
               DB      1                ;WR1  受信割り込み
               DB      1FH              ;割り込みイネーブル
               DB      4                ;WR4  CLOCK *16
               DB      47H              ;STOP bit 1  偶数パリティ
               DB      3                ;WR3  受信キャラクタ 7bit
               DB      61H              ;ｵｰﾄ･ｲﾈｰﾌﾞﾙ 受信ｲﾈｰﾌﾞﾙ
               DB      5                ;WR5  送信キャラクタ 7bit
               DB      0AAH             ;送信ｲﾈｰﾌﾞﾙ  RTS&DTR ON

SIO_B_TXBE:                             ;無効割り込み
SIO_B_ESTI:                             ;無効割り込み
SIO_B_RXCA:                             ;無効割り込み
SIO_B_RXSP:                             ;無効割り込み
               EI                       ;割り込みイネーブル
               RETI                     ;割り込みからの復帰

; チャネルA送信割り込み処理ルーチン
SIO_A_TXBE:    CALL    GET_TX_BYTE      ;送信データをバッファから得る
               JR      Z,SIO_A_TXBE_1   ;送信すべきデータがない
               OUT     (SIO_AD),A       ;送信データ・レジスタに書き込み
               JR      SIO_A_TXBE_2     ;送信割り込み処理完了
SIO_A_TXBE_1:  LD      A,28H            ;保留中の送信割り込みリセット
               OUT     (SIO_AC),A       ;コマンド・レジスタに書き込み
SIO_A_TXBE_2:  EI                       ;割り込みイネーブル
               RETI                     ;割り込みからの復帰

; チャネルA外部/ステータス割り込み処理ルーチン
SIO_A_ESTI:    LD      A,1              ;レジスタ・ポインタ・コマンド
               OUT     (SIO_AC),A       ;RR1を指定
               IN      A,(SIO_AC)       ;RR1を読み込む
               BIT     4,A              ;パリティ・エラーのチェック
               CALL    NZ,PARITY_ERROR  ;パリティ・エラー処理
               BIT     5,A              ;受信オーバーランのチェック
               CALL    NZ,OVERRUN_ERROR;オーバーラン・エラーの処理
               BIT     6,A              ;フレーミング・エラーのチェック
               CALL    NZ,FRAMING_ERROR;フレーミング・エラーの処理
               LD      A,10H            ;外部/ステータス割り込みリセット
               OUT     (SIO_AC),A       ;コマンド・レジスタへ書き込み
               EI                       ;割り込みイネーブル
               RETI                     ;割り込みからの復帰

; チャネルA受信割り込み処理ルーチン
SIO_A_RXCA:    IN      A,(SIO_AD)       ;受信データの読み込み
               CALL    PUT_RX_BYTE      ;受信バッファへの書き込み
               EI                       ;割り込みイネーブル
               RETI                     ;割り込みからの復帰

; チャネルA特殊受信状態割り込み処理ルーチン
SIO_A_RXSP:    LD      A,30H            ;エラー・リセット・コマンド
               OUT     (SIO_AC),A       ;コマンド・レジスタへ書き込み
               EI                       ;割り込みイネーブル
               RETI                     ;割り込みからの復帰
```

〈図10〉$\overline{RTS}$ や CTS 端子も使用した RS-232-C インターフェースの回路例

ず，チャネルBのWR_1とWR_2にも書き込みを行っていることです．

この理由は，SIOには割り込みベクタ・レジスタとステータス・アフェクツ・ベクタの設定ビットはチャネルBにしか存在しないため，たとえチャネルAしか使わない場合でも，割り込みを使う場合はチャネルBも初期化しなければならないことです．

● 最後に

SIOを使う際の基礎からレジスタの詳細，プログラミングについて解説しましたが，SIOは果てしなく高機能であるというのが実感です．モードによるプログラミングの違い，特に同期モードとHDLCモードについては機能が複雑で，ある順番にコマンドを実行し，レジスタを書き換えなければ動作しません．これらのプログラムは限られた紙面ではなかなかお伝えできません．そこでディスク頒布サービスでこれを行うこととしました．

このディスクには，SIOの非同期/同期/HDLCモードで実際にそのまま動作するプログラムと，

〈リスト5〉割り込みを使ったHDLCフレーム送受信プログラムの例

```
;       ＨＤＬＣ送受信プログラム

SIOAD   EQU     0       ;ＳＩＯチャネルAデータ・アドレス
SIOAC   EQU     1       ;ＳＩＯチャネルAコントロール・アドレス
SIOBD   EQU     2       ;ＳＩＯチャネルBデータ・アドレス
SIOBC   EQU     3       ;ＳＩＯチャネルBコントロール・アドレス

        ORG     0               ;ＲＯＭ開始アドレス
        DI                      ;割り込み禁止
        LD      SP,0            ;スタック・ポインタの設定
        IM      2               ;割り込みモード2
        LD      A,HIGH(INTVT)   ;割り込みベクタ上位8ビット
        LD      I,A             ;割り込みレジスタの設定
        JR      INISIO          ;ＳＩＯの初期化へジャンプ
;割り込みベクタ・テーブル
INTVT:  DEFW    0,0,0,0
        DEFW    TXBEI           ;送信バッファ・エンプティ
        DEFW    EXSTI           ;外部/ステータス割り込み
        DEFW    RXCAI           ;受信キャラクタ有効
        DEFW    SPRXI           ;特殊受信状態割り込み
;ＳＩＯの初期化
INISIO: LD      HL,SIOAP        ;チャネルAパラメータ・テーブル
        LD      B,13            ;チャネルAパラメータの個数
        LD      C,SIOAC         ;チャネルAコントロール・アドレス
        OTIR                    ;ＳＩＯチャネルAの初期化
        LD      HL,SIOBP        ;チャネルBパラメータ・テーブル
        LD      B,5             ;チャネルBパラメータの個数
        LD      C,SIOBC         ;チャネルBコントロール・アドレス
        OTIR                    ;ＳＩＯチャネルBの初期化
        JR      INIWRK          ;ワーク・エリアの初期化へ
;ＳＩＯチャネルA パラメータ
SIOAP:  DEFB    98H             ;ﾘｾｯﾄ・ﾁｬﾝﾈﾙ/CRCｼﾞｪﾈﾚｰﾀ
        DEFB    4               ;ｾﾚｸﾄWR4
        DEFB    20H             ;WR4=ｸﾛｯｸ1倍,HDLCﾓｰﾄﾞ
        DEFB    1               ;ｾﾚｸﾄWR1
        DEFB    13H             ;WR1=ｲﾈｰﾌﾞﾙTX/RX/STATUS割込
        DEFB    3               ;ｾﾚｸﾄWR3
        DEFB    0DDH            ;WR3=RX 8BITS,HUNT,RX CRC,ｻｰﾁ
        DEFB    6               ;ｾﾚｸﾄWR6
        DEFB    1               ;WR6=ｻｰﾁ・ｱﾄﾞﾚｽ=1
        DEFB    7               ;ｾﾚｸﾄWR7
        DEFB    7EH             ;WR7=ﾌﾗｸﾞ・ｼｰｹﾝｽ・ﾊﾟﾀｰﾝ
        DEFB    5               ;ｾﾚｸﾄWR5
        DEFB    0EBH            ;WR5=DTR,CCITT,RTS,TX CRC
;ＳＩＯチャンルB パラメータ
SIOBP:  DEFB    18H             ;ﾘｾｯﾄ・ﾁｬﾝﾈﾙ
        DEFB    1               ;ｾﾚｸﾄWR1
        DEFB    4               ;ｽﾃｰﾀｽ・ｱﾌｪｸﾂ・ﾍﾞｸﾀ・ｲﾈｰﾌﾞﾙ
        DEFB    2               ;ｾﾚｸﾄWR2
        DEFB    LOW(INTVT)      ;割り込みベクタ・テーブル下位8bit

;ワーク・エリアの初期化
INIWRK:         ;TXCF,TXDL,RXCF,RXDLを0に
                ;TXDPを送信ﾃﾞｰﾀ,RXDPを受信ﾃﾞｰﾀのｱﾄﾞﾚｽに設定
        EI                      ;割り込みイネーブル

;メイン・ルーチン
        ;
        ;  ～ 途中省略 ～
TXFRME: LD      HL,TXDB         ;送信データ・バッファ・アドレス
        LD      (TXDP),HL       ;送信データ・ポインタの設定
        LD      (HL),A          ;ｺﾝﾄﾛｰﾙ・ﾌｨｰﾙﾄﾞを送信バッファ書込
        XOR     A               ;Aレジスタをクリア
        LD      (TXCF),A        ;送信完了フラグをリセット
        LD      A,10            ;送信するデータのバイト数
        LD      (TXDL),A        ;STORE TRANSMIT DATA LENGTH
        LD      A,1             ;アドレス・フィールド
        OUT     (SIOAD),A       ;送信開始(送信割込イネーブル)
        LD      A,0C0H          ;送信アンダーランRESETコマンド
        OUT     (SIOAC),A       ;送信アンダーランのリセット
        ;  ～ 省略 ～

;送信バッファ・エンプティ割り込み処理ルーチン
TXBEI:  PUSH    AF              ;Aレジスタとフラグの退避
        PUSH    HL              ;ＨＬレジスタの退避
        LD      A,(TXDL)        ;残りの送信データのバイト数
        OR      A               ;ゼロかどうかの判定
        JR      Z,TXEND         ;残りの送信データがない
        DEC     A               ;残りの送信データのバイト数を-1
```

```
        LD      (TXDL),A        ;残りの送信データのバイト数保存
        LD      HL,(TXDP)       ;送信データのポインタ
        LD      A,(HL)          ;送信データをAレジスタに読込
        INC     HL              ;送信データ・ポインタをｲﾝｸﾘﾒﾝﾄ
        LD      (TXDP),HL       ;送信データ・ポインタを保存
        OUT     (SIOAD),A       ;データの送信
        JR      TXEXIT          ;送信割り込み処理から復帰
TXEND:  LD      A,28H           ;保留中の送信割り込みREST
        OUT     (SIOAC),A       ;保留中の送信割込RESET/FCS送信
        LD      A,1             ;送信完了フラグをセット
        LD      (TXCF),A        ;送信完了フラグを書き込み
TXEXIT: POP     HL              ;ＨＬレジスタを復旧
        POP     AF              ;Aレジスタとフラグを復旧
        EI                      ;割り込みイネーブル
        RETI                    ;割り込みからの復帰

;外部/ステータス割り込み処理ルーチン
EXSTI:  PUSH    AF              ;Aレジスタとフラグの退避
        IN      A,(SIOAC)       ;RR0を読み出し
        LD      (RR0),A         ;RR0を保存
        LD      A,10H           ;外部/ｽﾃｰﾀｽ割り込みﾘｾｯﾄ・ｺﾏﾝﾄﾞ
        OUT     (SIOAC),A       ;外部/ｽﾃｰﾀｽ割り込みのリセット
        POP     AF              ;Aレジスタとフラグの復旧
        EI                      ;割り込みイネーブル
        RETI                    ;割り込みからの復帰

;受信キャラクタ有効割り込み処理ルーチン
RXCAI:  PUSH    AF              ;Aレジスタとフラグの退避
        PUSH    HL              ;ＨＬレジスタを退避
        LD      A,(RXCF)        ;フレーム受信完了フラグを読込
        OR      A               ;受信フレームがすでにあるか?
        JR      NZ,RXEND        ;受信フレームがある場合は廃棄
        LD      A,(RXDL)        ;受信データのバイト数を読み込み
        INC     A               ;受信データのバイト数をｲﾝｸﾘﾒﾝﾄ
        JR      Z,RXEND         ;受信フレーム長が256バイトを超
        LD      (RXDL),A        ;受信データのバイト数を保存
        IN      A,(SIOAD)       ;受信ＦＩＦＯからのデータの読出
        LD      HL,(RXDP)       ;受信データ・バッファのアドレス
        LD      (HL),A          ;受信データをバッファに保存
        INC     HL              ;受信データのポインタをｲﾝｸﾘﾒﾝﾄ
        LD      (RXDP),HL       ;受信データのポインタを保存
        JR      RXEXIT          ;受信割り込み処理終了
RXEND:  IN      A,(SIOAD)       ;受信ＦＩＦＯのデータを読込廃棄
RXEXIT: POP     HL              ;ＨＬレジスタを復旧
        POP     AF              ;Aレジスタとフラグを復旧
        EI                      ;割り込みイネーブル
        RETI                    ;割り込みからの復帰

;特殊受信状態割り込み処理ルーチン
SPRXI:  PUSH    AF              ;Aレジスタとフラグを退避
        LD      A,1             ;ＲＲ1ポイント・コマンド
        OUT     (SIOAC),A       ;ＲＲ1を選択
        IN      A,(SIOAC)       ;ＲＲ1を読み込み（入力）
        LD      (RR1),A         ;ＲＲ1を保存
        BIT     7,A             ;フレームの終わりか?
        JR      Z,NOPAC         ;終わりでなければジャンプ
        LD      A,(RXDL)        ;受信フレームのバイト数を読込
        DEC     A               ;FCSが1バイト受信されるので無視
        LD      (RXDL),A        ;実際のフレームの内容を保存
        LD      A,1             ;受信完了フラグ
        LD      (RXCF),A        ;受信完了フラグをセット
NOPAC:  LD      A,30H           ;エラー・リセット・コマンド
        OUT     (SIOAC),A       ;エラー・リセット(割込許可)
        POP     AF              ;Aレジスタとフラグを復旧
        EI                      ;割り込みイネーブル
        RETI                    ;割り込みからの復帰
```

Z80SIOの機能をふんだんに使ったシリアル・ライン・モニタのプログラム，フレーム・レベルのHDLC送受信プログラムのサンプルが収められています．ご活用ください．

〈図11〉同期モード時のインターフェース回路例

◆参考文献◆

(1) VOLUME 1 DATABOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.
(2) Z80 Family User's Manual, ZILOG.

第10章

東芝TMPZ84C0xx/ザイログZ84Cxx/川崎製鉄KL5C8012の使い方

周辺組み込み＆Z80互換高速CPUの活用

高見　豊/菅原尚伸/菊地恭徳/北　孝

実際にマイコン・システムを設計する場合は，CPUとメモリだけでは役に立たず，必ずCTCやPIOなどのI/Oが必要になります．そこで，CPUと使用頻度の非常に高い周辺LSIを一つのICの中に入れてしまった，ワンチップ・マイコンというものがいくつか出ています．

ここではZ80CPUを核にして周辺LSIを組み込んだCPUについて取り上げてみます．

TMPZ84C0xxシリーズ

もともとTMPZ84Cシリーズは，東芝製のZ80ファミリを示す型番でしたが，最近ではTMPZ84C015に代表される，Z80CPUを核にZ80ファミリを内蔵したシリーズが有名です．

表1にTMPZ84Cシリーズを示します．周辺LSIの組み合わせで，いくつかシリーズがあります．ここではよく使われるC015，C013，C011について解説します．

TMPZ84C015

● ピン配置，ピン名称

ピン配置を図1に示します．基本的にZ80CPUおよびCTCやPIO，SIOと同じ名前のピンは，いままで説明してきた単体のCPUや周辺LSIの動作とまったく同じです．ただしTMPZ84C015で拡張された機能などのために，特有の信号ピンがいくつかあります(**表2**)．

〈表1〉TMPZ84Cシリーズ

型番	構成
TMPZ84C011AF	Z80CPU＋CGC＋CTC＋I/Oポート
TMPZ84C013AF	Z80CPU＋CGC＋CTC＋SIO＋WDT
TMPZ84C015AF	Z80CPU＋CGC＋CTC＋PIO＋SIO＋WDT
TMPZ84C810AF	Z80CPU＋CGC＋CTC＋I/Oポート＋SIO＋DMA＋MMU＋WDT
TMPZ84C710A	Z80CPUをコントローラとしたISDN基本インターフェース用チップ

CGC：クロック・ジェネレータ・コントローラ，
WDT：ウォッチドグ・タイマ，
MMU：メモリ・マネージメント・ユニット

TMPZ84C015ではCGC(クロック・ジェネレータ・コントローラ)を内蔵しているので，$XTAL_1$と$XTAL_2$間に図2のように水晶振動子とコンデンサを接続するだけで，クロックを発生させることができます．ここに接続する水晶発振子は，動作させたいクロック周波数の2倍の周波数のものを接続します．

IEOとIEIは，Z80ファミリLSIの割り込み優先順位の制御線です．内蔵のI/O以外に，さらに外付けにZ80ファミリを接続し，割り込みを使用するときに接続が必要になります．外付けにI/Oを増設しないときや，増設したとしてもZ80ファミリではないときは，IEIをプルアップしておきます．IEOは解放でかまいません．

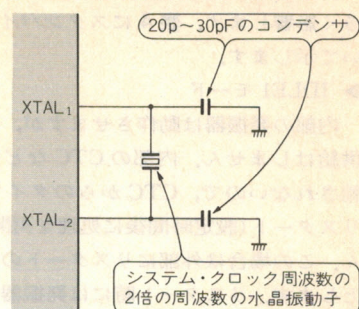
CLKINは外部クロック入力端子で，内蔵CGCを使用せずに外部からクロックを入力するときに使用します．単体Z80CPUのCLK端子と考えてください．通常はCLKOUTに接続します．

CLKOUTは内蔵CGCで2分周されたシステム・クロックの出力端子です．さらにTMPZ84C015には，内蔵のCPUを切り離すためのEV端子があります．これはICEなどのツールを使ってシステムを開発するときに使用する端子で，通常はプルダウンしておきます．またICT端子はICテスト用の端子なので解放のままにしておきます．

● Z80ファミリLSIとの違い

もう一つ注意すべき点は，Z80CTCではパッケージのピン数の関係で省略されていたチャネル3のZC/TO_3の出力ピンがある点，そしてZ80SIOのボンディ

〈図1〉TMPZ84C015のピン配置

〈図2〉水晶発振子の接続例

〈表2〉TMPZ84C015特有の信号ピン

ピン名称	入出力	機能
XTAL1	入力	水晶発振子用接続端子
XTAL2	出力	
IEO	出力	割り込みイネーブル出力
IEI	入力	割り込みイネーブル入力
CLKIN	入力	外部クロック入力
CLKOUT	出力	クロック出力
EV(I)	入力	エバリュエータ用信号
A7RF	出力	補助アドレス・ビット
WDTOUT	出力	ウォッチドグ・タイマ出力
ICT	入力	テスト用端子

ング・オプションのような制約がなく，チャネルBの送受信クロック入力やSYNC入力，DTR出力がすべて出ている点です．

このためソフトウェア的にはまったく同じといっても，TMPZ84C015で作った回路を，そのまま単品のCPUやCTC，SIOの組み合わせに置き換えようとしたとき，信号ピンが出ていないなどの問題がおこるので注意します．

さらにZ80では7ビット分しか有効ビットがなかったリフレッシュ・アドレスですが，TMPZ84C015では，A7RF端子を装備することで，8ビット分のリフレッシュ・アドレスを出力することが可能です．

● 内蔵I/Oついて

表3にTMPZ84C015内蔵のI/Oのアドレスを示します．イメージ・アドレスはなくフル・デコードされています．これはCPU内部で固定されているので，外部にI/Oを増設するときは，同じアドレスに重ならないようにアドレス・デコード回路を設計してください．アドレスがすでに決まっているだけで，機能は単品のZ80ファミリとまったく同じです．

独自のI/Oポートとしては，ウォッチドグ・タイマやスタンバイ・モード，内蔵I/Oの割り込み優先順位設定などのポートが追加されています．

● スタンバイ機能とは

TMPZ84CはCMOSのため，それ自体の消費電力も小さく，またROMやRAMにもCMOSの物を使えば，マイコン回路の消費電力はほとんどゼロに近くなります．

しかしクロック回路などで発振が続いていると，発振回路の消費電力だけでなくCPUやROMなどでも電力を消費し，電池動作の機器などでは思ったように電池寿命が延びません．

そこでTMPZ84Cでは，HALT命令を実行するとスタンバイ機能と呼ばれる低消費電力モードに入ります．さらにクロックを止めてしまえば消費電力はほとんどゼロになります．

TMPZ84Cでは，HALT命令実行時のクロックに関する動作を4通り設定することが可能です．この

HALTモードの設定はCPUの動作や消費電力に大きく影響します．表4にスタンバイ・モード設定について示します．

▶ IDLE1 モード

内部の発振器は動作させますが，外部へのクロック供給はしません．内部のCTCなどにもクロックは供給されないので，CTCからのタイマ割り込みによるリスタート(設定時間後に処理を再開する)はできません．この場合は外部にリスタートのための回路を必要とします．リスタート時には発振器のウォーミング・アップが必要ないので，素早く処理を再開させることができますが，発振器が動作しているために，数mAの電流を消費します．

▶ IDLE2 モード

内部発振器を動作させ，さらに外部へのクロック供給をします．内部のCTCなどにもクロックは供給されるので，CTCからの割り込みによるリスタート(設定時間後に処理を再開する)ができます．しかし，CTCが動作しているために，消費電流は通常の半分程度にしかなりません．

▶ STOP モード

内部動作をすべて停止するので，消費電流はほとんどゼロになります．しかしリスタート時にはウォーミング・アップを行うので，処理の再開には2^{14}クロック・サイクルの時間がかかります．

なお，外部クロックを使用すると，HALT命令を実行してもSTOPモードにはなりません．

▶ RUN モード

HALT命令を実行してもスタンバイ・モードにはなりません．したがって消費電流も通常の動作時と同じになります．つまりZ80と同様に，内部的にはNOP命令の実行を繰り返すモードです．

● スタンバイ・モードの設定

システム設定に関する重要な設定なので，簡単には変更できないようなしかけがされています．まずホールト・モード設定レジスタ(アドレスF1h)に対してDBhを書き込みます．それからホールト・モード・コントロール・レジスタ(アドレスF0h)のビット4，3で，ホールト命令実行時のスタンバイ・モードを設定します．またこのレジスタの上位ビットは，後述するウォッチドグ・タイマのレジスタになっているので，ウォッチドグ・タイマでの設定も考えてデータを書き込みます．

● ウォッチドグ・タイマとは

マイコンの処理は，いくつかある処理を決められた順序で繰り返すという動作が大半をしめていると思われます．

また本来ハードウェアやソフトウェアが正常であれば，暴走などという状態にはならないはずです．しかし組み込み用マイコンなどの場合は，置かれる環境などにより外部からノイズや予期しない信号で，誤動作することが考えられます．

このような状態になったときに，自動的にリセットをかけられるシステムがあると，ふたたび決められた処理を繰り返す動作に復帰でき便利です．

これを実現するのがウォッチドグ・タイマです．一定時間以内にプログラムがカウンタの値をクリアしないと，リセットに接続された出力がアクティブになり，システムをリセットするというものです．よってソフトウェアで，その一定時間以内にカウンタをクリアするようにしなければなりません．

● ウォッチドグ・タイマの設定

ウォッチドグ・タイマ設定レジスタには，ウォッチドグ機能を使うか使わないかのイネーブル設定ビットと，カウンタをクリアしてからリセットがかかるまでの時間を選択，設定します．

リセットまでの時間は，システム・クロック×2^{16}，2^{18}，2^{20}，2^{22}の4種類が選択できます．システム・クロック×2^{16}を設定した場合，クロック10 MHzでは6.5 msの周期となります．1命令の実行クロックを平均10クロックとすると，約6500ステップ分のプログラムの実行時間に相当します．

またスタンバイ・モードの設定で説明したように，

〈表3〉TMPZ84C015内蔵I/Oのポート・アドレス

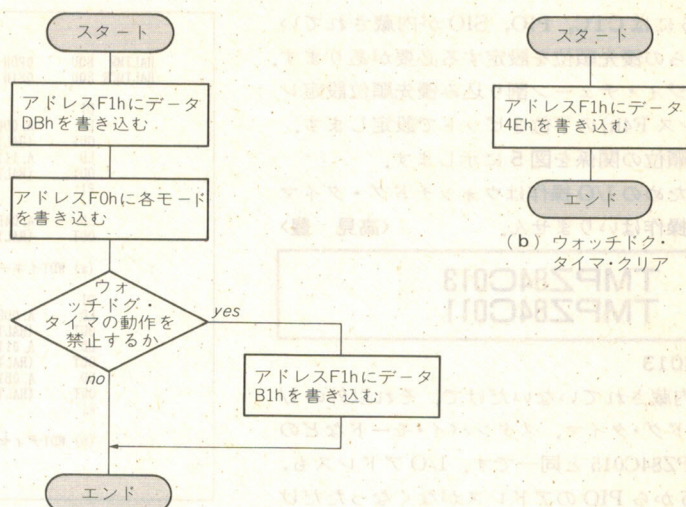
内蔵I/Oデバイス	チャネル	I/Oアドレス
CTC	チャネル0	10h
	チャネル1	11h
	チャネル2	12h
	チャネル3	13h
SIO	チャネルA　データ	18h
	チャネルA　コマンド/ステータス	19h
	チャネルB　データ	1Ah
	チャネルB　コマンド/ステータス	1Bh
PIO	チャネルA　データ	1Ch
	チャネルA　コマンド	1Dh
	チャネルB　データ	1Eh
	チャネルB　コマンド	1Fh
ウォッチドグ・タイマ/スタンバイ・モード設定		F0h
ウォッチドグ・タイマ・コマンド・レジスタ		F1h
デイジィ・チェーン割り込み優先順位設定レジスタ		F4h

〈表4〉スタンバイ・モードのいろいろ

モード	CGC発振器	CPU	CTC	PIO	SIO	WDT	CLK OUT端子
IDLE1	○	×	×	×	×	×	×
IDLE2	○	×	○	×	×	×	○
STOP	×	×	×	×	×	×	×
RUN	○	○	○	○	○	○	○

○：動作継続，×：動作停止

〈図3〉 スタンバイ・モードとウォッチドグ・タイマの設定

〈図4〉
スタンバイ・モードとウォッチドグ・タイマの設定の流れ

(a) スタンバイ・モード/ウォッチドグ・タイマの設定

(b) ウォッチドグ・タイマ・クリア

ウォッチドグ・タイマのレジスタは二重コントロールになっているので同様の手順でレジスタを設定します．

スタンバイ・モードとウォッチドグ・タイマの設定を図3に，設定や解除の流れを図4に示します．またリスト1に設定プログラム例を示します．

● ウォッチドグ・タイマの活用

ウォッチドグ・タイマとはマイコンの信頼性を上げるためのものです．ではウォッチドグ・タイマをクリアするためのプログラムは，どのように埋め込んだらよいのでしょうか．

決められた時間以内にクリアするわけですから，タイマ割り込みなどで処理すると簡単そうですが，ウォッチドグ・タイマをタイマ割り込みでクリアするのはナンセンスです．

ウォッチドグ・タイマとは，メイン・ルーチンが，正しく繰り返し実行されているかをチェックするものと考えてください．よってメイン・ルーチンのループの中の1箇所だけに入れるべきです．しかもサブルーチンなどにせずに直接OUT命令を実行したほうがよいでしょう．

また，例えばシステム・クロックが10 MHzのとき，ウォッチドグ・タイマの時間は最長でもシステム・クロック×2^{22}時間，約0.4秒となります．つまりメイン・ルーチンを1回ループするときの処理時間は，この時間以内に収まるようにしなければなりません．

例えば工作機械のアームなどは，動き出してから0.5秒もあれば作業員を跳ね飛ばすこともできるでしょう．つまり0.5秒もの長時間(マイコンにとっては長時間と考えるべき)マイコンが暴走していたらたいへんなことになる可能性もあるのです．

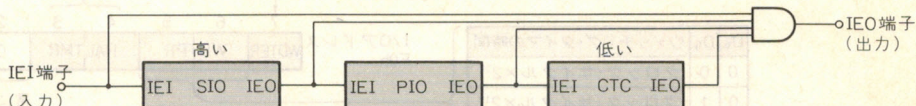
● 割り込み優先順位

第6章で解説したように，Z80はデイジィ・チェー

優先順位 高←低	D_2	D_1	D_0
CTC-SIO-PIO	0	0	0
SIO-CTC-PIO	0	0	1
CTC-PIO-SIO	0	1	0
PIO-SIO-CTC	0	1	1
PIO-CTC-SIO	1	0	0
SIO-PIO-CTC	1	0	1

(a) 割り込み優先順位の設定

(b) "101"を書き込んだときの例

〈図5〉
TMPZ84C015の割り込み優先順位の設定

ン方式により割り込み優先順位を制御しています. TMPZ84C015にはCTC, PIO, SIOが内蔵されているので, これらの優先順位を設定する必要があります.

これはデイジィ・チェーン割り込み優先順位設定レジスタ(アドレスF4h)の下位3ビットで設定します. 設定値と優先順位の関係を図5に示します.

この設定のためのI/O操作はウォッチドグ・タイマのような二重操作はいりません. 〈高見 豊〉

〈リスト1〉 ウォッチドグ・タイマ設定プログラム例

```
HALTMR  EQU    0F0H        ;ホールト・モード・レジスタI/O
HALTMCR EQU    0F1H        ;コマンド・レジスタI/O

        DI
        LD     A,0DBH      ;ホールト・モード解除データ
        OUT    (HALTMCR),A ;書き込み
        LD     A,11111011B ;WDTイネーブル  RUNモード
        OUT    (HALTMR),A
        EI

        LD     A,04EH      ;WDTクリア・データ
        OUT    (HALTMCR),A ;WDTクリア

;       (a) WDTイネーブル・プログラム

        DI
        LD     A,0DBH      ;ホールト・モード解除データ
        OUT    (HALTMCR),A ;書き込み
        LD     A,01111011B ;WDTディセーブル  RUNモード
        OUT    (HALTMR),A
        LD     A,0B1H      ;WDTディセーブル・データ
        OUT    (HALTMCR),A ;書き込み
        EI

;       (b) WDTディセーブル・プログラム
```

TMPZ84C013
TMPZ84C011

● TMPZ84C013

Z80PIOが内蔵されていないだけで, それ以外のI/Oやウォッチドグ・タイマ, スタンバイ・モードなどの設定も, TMPZ84C015と同一です. I/Oアドレスも, TMPZ84C015からPIOのアドレスがなくなっただけで, 他のI/Oアドレスに変更はありません. ピン配置を図7と表5に示します.

● TMPZ84C011

これはTMPZ84C015からSIOとWDTをとり, さらに8ビット・パラレル入出力ポートを5チャネル装備した, パラレル入出力を強化したCPUです. 図7と表5にピン配置を表6に内蔵I/Oアドレスを示します. またCTCの出力である ZC/TO_3 もありません.

このパラレル・ポートはZ80PIOとは異なり, Z80PIOの割り込みを使わないモード3のビット・モード専用のパラレル・ポートと同じような使い方をします.

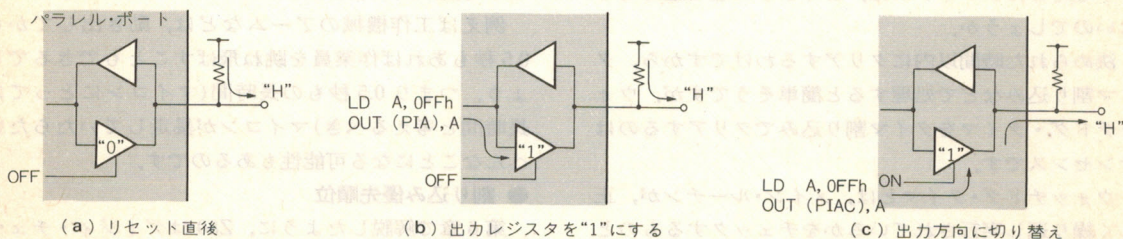
● パラレル・ポートの使い方

パラレル・ポートの入出力は1ビット単位で設定でき, 対応するチャネルの入出力設定レジスタの対応するビットを設定します. "0" で入力, "1" で出力です.

〈図6〉 パラレル・ポートの動作

(a) リセット直後

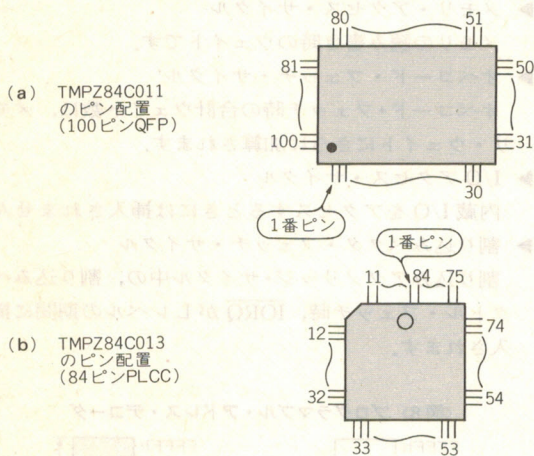
(b) 出力レジスタを"1"にする

(c) 出力方向に切り替え

〈表5〉TMPZ84C013/011のピン配置

ピン名称	TMPZ84C011	TMPZ84C013
D_0 〜 D_7	6 〜 13	83 〜 76
A_0 〜 A_{15}	29 〜 44	1 〜 16
PA_0 〜 PA_7	68 〜 61	—
PB_0 〜 PB_7	76 〜 79	—
PC_0 〜 PC_7	60 〜 53	—
PD_0 〜 PD_5	96 〜 91	—
PD_6	89	—
PD_7	88	—
PE_0 〜 PE_7	85 〜 78	—
$\overline{\text{M1}}$	25	17
$\overline{\text{RD}}$	19	30
$\overline{\text{WR}}$	20	20

ピン名称	TMPZ84C011	TMPZ84C013
CLK/TRG_0 〜 CLK/TRG_3	5 〜 2	75 〜 72
ZC/TO_0 〜 ZC/TO_2	99 〜 97	68 〜 70 (ZC/TO_3 : 71)
IEI	1	60
IEO	87	59
$XTAL_1$	49	63
$XTAL_2$	48	64
$\overline{\text{MREQ}}$	17	23
$\overline{\text{IORQ}}$	18	21
$\overline{\text{WAIT}}$	22	19
$\overline{\text{BUSREQ}}$	23	18
$\overline{\text{BUSACK}}$	21	29
$\overline{\text{HALT}}$	16	31
$\overline{\text{RFSH}}$	26	26
EV	15	58
TEST	86	—
ICT	—	42, 44
MS_1	47	—
MS_2	46	—
CLK	52	66 (CLKOUT)

ピン名称	TMPZ84C011	TMPZ84C013
CLKIN	—	67
$\overline{\text{RESET}}$	24	28
$\overline{\text{INT}}$	14	25
$\overline{\text{NMI}}$	45	56
$\overline{W/RDY_A}$	—	32
$\overline{W/RDY_B}$	—	54
$\overline{SYNC_A}$	—	33
$\overline{SYNC_B}$	—	53
RxD_A	—	34
RxD_B	—	52
$\overline{RxC_A}$	—	35
$\overline{RxC_B}$	—	51
$\overline{TxC_A}$	—	36
$\overline{TxC_B}$	—	50
TxD_A	—	37
TxD_B	—	49
$\overline{DTR_A}$	—	38
$\overline{DTR_B}$	—	48
$\overline{RTS_A}$	—	34
$\overline{RTS_B}$	—	47
$\overline{CTS_A}$	—	40
$\overline{CTS_B}$	—	46
$\overline{DCD_A}$	—	41
$\overline{DCD_B}$	—	45
V_{CC}	27, 51, 90	43, 84
V_{SS}	28, 50, 77, 100	22, 62

〈図7〉TMPZ84C011と013のピン配置

〈表6〉TMPZ84C011内蔵I/Oのポート・アドレス

内蔵I/Oデバイス	チャネル	I/Oアドレス
CTC	チャネル0	10h
	チャネル1	11h
	チャネル2	12h
	チャネル3	13h
パラレルI/O	チャネルA入出力設定レジスタ	54h
	チャネルB入出力設定レジスタ	55h
	チャネルC入出力設定レジスタ	56h
	チャネルD入出力設定レジスタ	34h
	チャネルE入出力設定レジスタ	44h
	チャネルA データ	50h
	チャネルB データ	51h
	チャネルC データ	52h
	チャネルD データ	30h
	チャネルE データ	40h

リセット時はすべてのポートの設定が入力になっています．またこのパラレル・ポートは，ポートが入力に設定されていても，書き込み動作をすると出力レジスタにデータがラッチされます．それからポートを出力に設定すると，書き込んだ値が即座に出力されるようになっています．

よって図6のように端子をプルアップしておくと，リセット直後はポートが入力に設定されるので，端子の状態としてはHレベルを出力することになります．

まずCPUがリセットされるとPIOの出力レジスタはクリアされ“0”になります．このあと“1”を書き込むと出力レジスタにラッチされ，次に方向を出力に設定しても端子の状態はHレベルを保ちます．これにより，Z80PIOや8255などで問題になる，初期化時に瞬間的にLレベルが出力されるという問題がありません．

Z84C15/Z84C11

Z84Cxx シリーズ(ザイログ)は，TMPZ84C015(東芝)に対応する Z84C15 と，TMPZ84C011 に対応する Z84C11 の 2 種類がよく使われます．内蔵 I/O の種類，アドレス，設定方法などは，Z84C15はTMPZ84C015 と，Z84C11 は TMPZ84C011 とまったく同じです．

しかしこの 2 種類には，次の機能が追加されています．

▶ パワー ON リセット

▶ プログラマブル・ウェイト・ステート・ジェネレータ

▶ プログラマブル・アドレス・デコーダ

▶ 32 ビット CRC ジェネレータ・チェッカ(SIO チャネル A のみ)

▶ SIO 送受信クロック・シュミット・トリガ入力

▶ CGC 出力周波数 1/1，1/2 の選択が可能

▶ EV モード時，外部回路不要

また当然のことながら，Z84C11 には SIO が内蔵されていないので，32 ビット CRC や送受信クロック入力のシュミット・トリガ機能はありません．またプログラマブル・アドレス・デコーダの機能もありません．

PIO，CTC，SIO，WDT のアドレスは，TMPZ84C015 や 011 ととまったく同じですが，これ以外に，上記の機能をコントロールするための，システム・コントロール・レジスタが追加されています(**表 7**)．

ここではとくに，パワー ON リセット機能と，プログラマブル・ウェイト・ステート・ジェネレータ，プログラマブル・アドレス・デコーダについて説明します．

● パワー ON リセット

Z84C15/C11 のリセット端子はオープン・ドレイン端子で，切り離し可能なリセット IC が内蔵された端子と考えることができます．電源投入時は，内蔵のリセット IC がイネーブルに設定されるので，電源電圧が約 2.2 V を越えてから約 25～75 ms の期間，L レベルを出力します．また立ち下がりエッジを検出すると，システム・クロックで 16 クロックの時間，L レベルを出力します．

またシステム・コントロール・レジスタを設定することで，リセット IC をディセーブルに設定することも可能です．この場合は通常の Z80CPU と同様，システム・クロックで 3 クロック以上の時間，リセット端子に L レベルを入力することで，リセットをかけることができます．

● ウェイト・ステート・ジェネレータ

Z84C15/C11 は，各サイクルに外付け回路なしでウェイトを追加することができます．

このとき，$\overline{\mathrm{WAIT}}$ 端子によるウェイト・コントロールは，ウェイト・ステート・ジェネレータによるウェイト・ステート終了後にサンプリングされます(割り込みアクノリッジ時を除く)．

以下のサイクルのときにウェイトを入れることができます．

▶ メモリ・アクセス・サイクル

メモリの読み書き時のウェイトです．

▶ オペコード・フェッチ・サイクル

オペコード・フェッチ時の合計ウェイト数は，メモリ・ウェイトにさらに加算されます．

▶ I/O アクセス・サイクル

内蔵 I/O をアクセスするときには挿入されません．

▶ 割り込みベクタ・フェッチ・サイクル

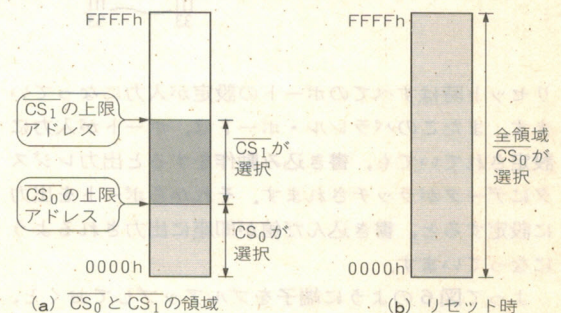
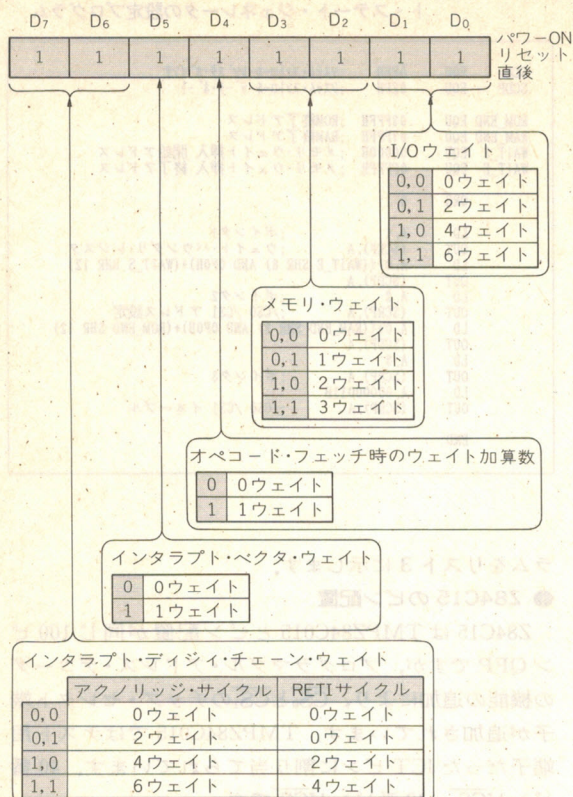
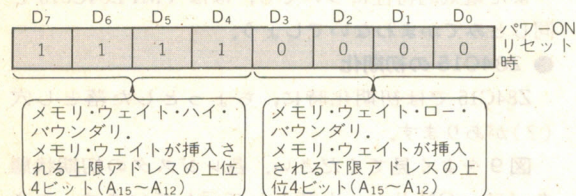
割り込みアクノリッジ・サイクル中の，割り込みベクトル・フェッチ時，$\overline{\mathrm{IORQ}}$ が L レベルの期間に挿入されます．

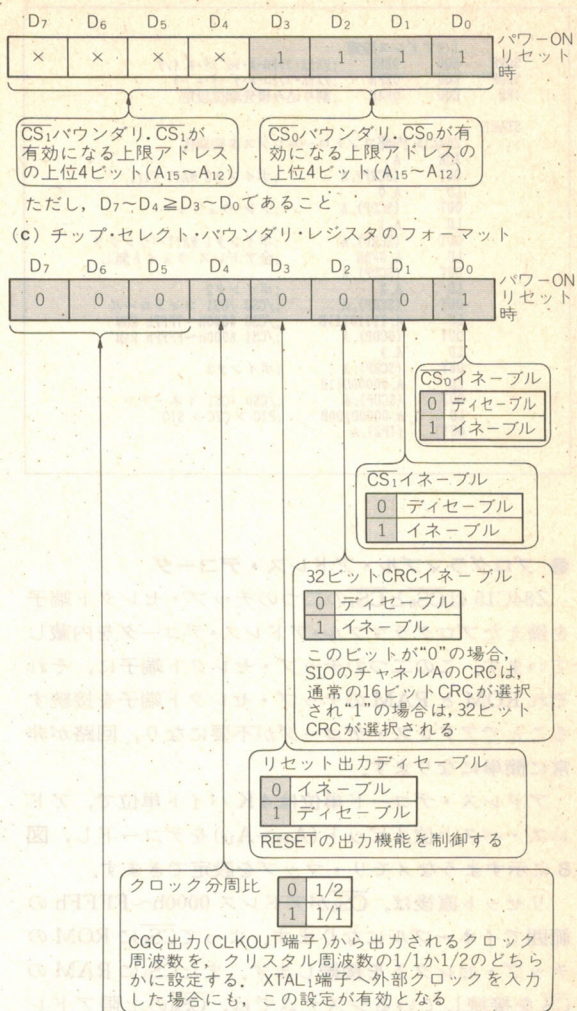
〈表 7〉 Z84C15/C11 に追加されたシステム・コントロール・レジスタの I/O アドレス

SCRP(システム・コントロール・レジスタ・ポインタ)	EEh
SCDP(システム・コントロール・データ・ポート)	EFh

〈表 8〉 コントロール・レジスタのアドレス

00000000b(00h)	WCR(ウェイト・コントロール・レジスタ)
00000001b(01h)	MWBR(メモリ・ウェイト・バウンダリ・レジスタ)
00000010b(02h)	CSBR(チップ・セレクト・バウンダリ・レジスタ)
00000011b(03h)	MCR(Misc コントロール・レジスタ)

〈図 8〉 プログラマブル・アドレス・デコーダ

〈図9〉 各種システム・コントロール・レジスタの設定

(a) ウェイト・コントロール・レジスタのフォーマット

(b) メモリ・ウェイト・バウンダリ・レジスタのフォーマット

(c) チップ・セレクト・バウンダリ・レジスタのフォーマット

(d) Miscコントロール・レジスタのフォーマット

▶割り込みアクノリッジ・サイクル

割り込みアクノリッジ・サイクルにおいて，$\overline{M1}$がLレベルになってから，$\overline{IORQ}$がLレベルを出力するまでの間に挿入されます．さらに，RETI(割り込み復帰)サイクル中にも挿入されます．これらは，デイジィ・チェーン接続された周辺LSIへのアクセス時間を確保するために挿入されます．

リセット後はすべてのサイクルに，最大数のウェイトが自動的に挿入されます．そして16回目のオペコード・フェッチ・サイクルから，ウェイト・ステートの自動挿入はなくなります．

よって，アクセス速度の遅いメモリなどを使用する場合は，リセット後16ステップ以内に，ウェイト・ステート・ジェネレータに対して各サイクルのウェイト数を設定しなければなりません．

また，メモリ・アクセス・サイクルに対するウェイトは，ウェイトを入れるアドレス範囲を指定できます．つまり，RAMには高速なメモリを使っているが，ROMが遅いという場合，ROM領域だけにウェイトを入れることができます．

アクセス速度とメモリ・アクセス時のウェイト数の目安としては，システム・クロックが6 MHzのCPUの場合，200 ns以上のメモリを使用するなら1ウェイト，それより速いメモリならウェイトの必要はありません．システム・クロック10 MHzのCPUの場合，200 ns以上のメモリなら2ウェイト，200 ns～100 nsなら1ウェイト，100 ns以下ならウェイトなしとなります．

〈リスト2〉システム・コントロール・レジスタの設定例

```
;         I/Oアドレス設定
SCRP      EQU       0EEH      ;システム・コントロール・レジスタ・ポインタ
SCDP      EQU       0EFH      ;システム・コントロール・データ・ポート
IPR       EQU       0F4H      ;割り込み優先順位設定

START:
;         Z84C15 内部コントロールレジスタ初期化
          XOR       A
          OUT       (SCRP),A            ;ポインタ0 WAIT Ctrl
          LD        A,0
          OUT       (SCDP),A            ;ノン・ウェイト
          LD        A,1
          OUT       (SCRP),A            ;ポインタ1 WAITバウンダリ
          LD        A,0F0H              ;全アドレス ウェイト無し
          OUT       (SCDP),A
          LD        A,2                 ;ポインタ2
          OUT       (SCRP),A            ;/CS0 /CS1 コントロール
          LD        A,11110111B         ;/CS0 0000h～7FFFh ROM
          OUT       (SCDP),A            ;/CS1 8000h～FFFFh RAM
          LD        A,3
          OUT       (SCRP),A            ;ポインタ3
          LD        A,00000011B
          OUT       (SCDP),A            ;/CS0 /CS1 イネーブル
          LD        A,00000100B         ;PIO > CTC > SIO
          OUT       (IPR),A
```

〈リスト3〉プログラマブル・アドレス・デコーダとウェイト・ステート・ジェネレータの設定プログラム

```
SCRP     EQU      0EEH     ;システム・コントロール・レジスタ・ポインタ
SCDP     EQU      0EFH     ;システム・コントロール・データ・ポート

ROM_END EQU       03FFFH   ;ROM終了アドレス
RAM_END EQU       07FFFH   ;RAM終了アドレス
WAIT_S   EQU      00000H   ;メモリ・ウェイト挿入 開始アドレス
WAIT_E   EQU      0FFFFH   ;メモリ・ウェイト挿入 終了アドレス

         ORG      0

         LD       A,1                ;ポインタ1
         OUT      (SCRP),A           ;ウェイト・バウンダリ・レジスタ
         LD       A,0+((WAIT_E SHR 8) AND 0F0H)+(WAIT_S SHR 12)
         OUT      (SCDP),A
         LD       A,2                ;ポインタ2
         OUT      (SCRP),A           ;/CS0 /CS1 アドレス設定
         LD       A,0+((RAM_END SHR 8) AND 0F0H)+(ROM_END SHR 12)
         OUT      (SCDP),A
         LD       A,3
         OUT      (SCRP),A           ;ポインタ3
         LD       A,00000011B
         OUT      (SCDP),A           ;/CS0 /CS1 イネーブル

         END
```

● プログラマブル・アドレス・デコーダ

Z84C15は$\overline{CS_0}$と$\overline{CS_1}$の二つのチップ・セレクト端子を備えたプログラマブル・アドレス・デコーダを内蔵しています．この二つのチップ・セレクト端子に，それぞれROMとRAMのチップ・セレクト端子を接続することでアドレス・デコーダが不要になり，回路が非常に簡単になります．

アドレス・デコード単位は4Kバイト単位で，アドレス・バス上位4ビット(A_{15}～A_{12})をデコードし，図8に示すようなメモリ・マップを設定できます．

リセット直後は，$\overline{CS_0}$がアドレス0000h～FFFFhの範囲でイネーブルになります．よって$\overline{CS_0}$にROMのチップ・セレクトを接続します．また$\overline{CS_1}$にRAMの$\overline{CS}$を接続しているシステムでは，$\overline{CS_0}$の上限アドレスを設定し，$\overline{CS_1}$をイネーブルにしないと，RAMが選択されません．

● システム・コントロール・レジスタの使い方

プログラマブル・ウェイト・ステート・ジェネレータやアドレス・デコータなどの設定は，システム・コントロール・レジスタをアクセスすることで設定します．

システム・コントロール・レジスタは，SCRP(システム・コントロール・レジスタ・ポインタ：I/OアドレスEEh)にアクセスしたいコントロール・レジスタのアドレスを指定し，SCDP(システム・コントロール・データ・ポート：I/OアドレスEFh)で読み書きするという手順を取ります．

各種コントロール・レジスタのアドレスとコントロール・データの意味を**表8**と**図9**に，システム・コントロール・レジスタの設定方法の例を**リスト2**に示します．またプログラマブル・アドレス・デコーダとメモリ・アクセスにウェイトを入れる領域の設定を，アセンブラの算術演算子を使って自動的に計算するプログラムを**リスト3**に示します．

● Z84C15のピン配置

Z84C15はTMPZ84C015とピン配置が同じ100ピンQFPですが，プログラマブル・アドレス・デコーダの機能の追加により，$\overline{CS_0}$と$\overline{CS_1}$のチップ・セレクト端子が追加されています．TMPZ84C015ではテスト用端子だったICTピンに割り当てられています．40番ピンが$\overline{CS_0}$，42番ピンが$\overline{CS_1}$です．

また電気的特性についても，ほぼTMPZ84C015と同じとみてかまわないでしょう．

● Z84C15の初期化

Z84C15では初期化時に，ちょっとした落とし穴(？)があります．

図9をよく見てください．各レジスタの初期状態をパワーONリセット時として示しています．つまり電源を投入したときにしかクリアされないのです．

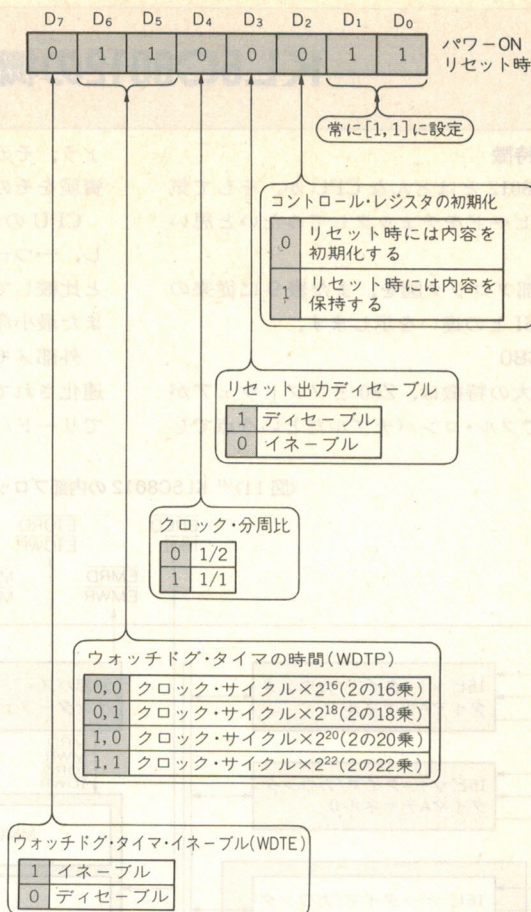
ということは，最初に電源を投入してこれらのレジスタを一度だけ設定すれば，次にリセット・スイッチによりリセットしても，その設定がクリアされずに残るわけです．

これでよく失敗するのは，プログラムの開発中などのときです．一度初期化がうまくいくと，その設定がリセット後も残りますから，その後のプログラムの修正で，実は初期化部分を誤って変えてしまっても，リセット・スイッチによるリセットではそのまま動作してしまうわけです．

こうして完成したと思ってROM化し，改めて電源ONで走らせると…ということになりかねません．注意しましょう．

● 1/2分周と1/1分周クロック

これは例えば，5MHz版のCPUに10MHzの水晶振動子を接続し，普段は1/2分周の5MHzで動作

〈図 10〉
Z84C11 のウォッチドグ・タイマ・レジスタ
(I/O アドレス：F0h)

させ，ターボ・モードで 1/1 分周の 10 MHz にできる…というものではありません．

1/1 分周を使うときは，クロック周波数と同じ周波数の水晶振動子を使い，少しでも低消費電力にしたいときに半分のクロック周波数で動作させ，処理速度を優先させたいときに 1/1 分周に設定するという目的で使います．

よって 1/1 分周に設定したときのクロック周波数が，CPU の最高動作クロック周波数を超えないように周波数を選択します．

● Z84C11 のピン配置

これもピン配置は TMPZ84C011 と同じですが，ウォッチドグ・タイマ機能が追加されているので，その出力端子である $\overline{\text{WDTOUT}}$ 端子が，パラレル・ポートのチャネル E のビット 7(78 番ピン)のピンと兼用に割り当てられています．

ウォッチドグ・タイマがイネーブルに設定されると，$\overline{\text{WDTOUT}}$ になります．このときチャネル E のビット 7 に書き込まれた値は，$PE_7/\overline{\text{WDTOUT}}$ ピンには影響は与えませんが，出力ラッチには書き込まれています．また読み出しをしたときは，$\overline{\text{WDTOUT}}$ の出力状態が読み込まれます．リセット時は，ウォッチドグ・タイマはディセーブルです．

またプログラマブル・アドレス・デコーダなどの機能がないので，システム・コントロール・レジスタの CSBR と MCR はありません．またこのため内蔵のリセット IC のイネーブル・コントロールや，CGC クロックの分周比の設定のビットが，モード設定レジスタ(アドレス F0h)に割り当てられているので注意します(図 10)．

また図 10 でわかるように，これも C15 と同様に，リセット時にコントロール・レジスタの状態を保持することもできます．しかしプログラマブルに設定できる点が違います．

クロック周波数の分周設定についても C15 と同様です．

〈菅原尚伸〉

KL5C8012の概要

● KL5C8012 の特徴

ここでは KL5C8012 とはどんな CPU か，そして気になるその処理スピードをチェックしてみたいと思います．

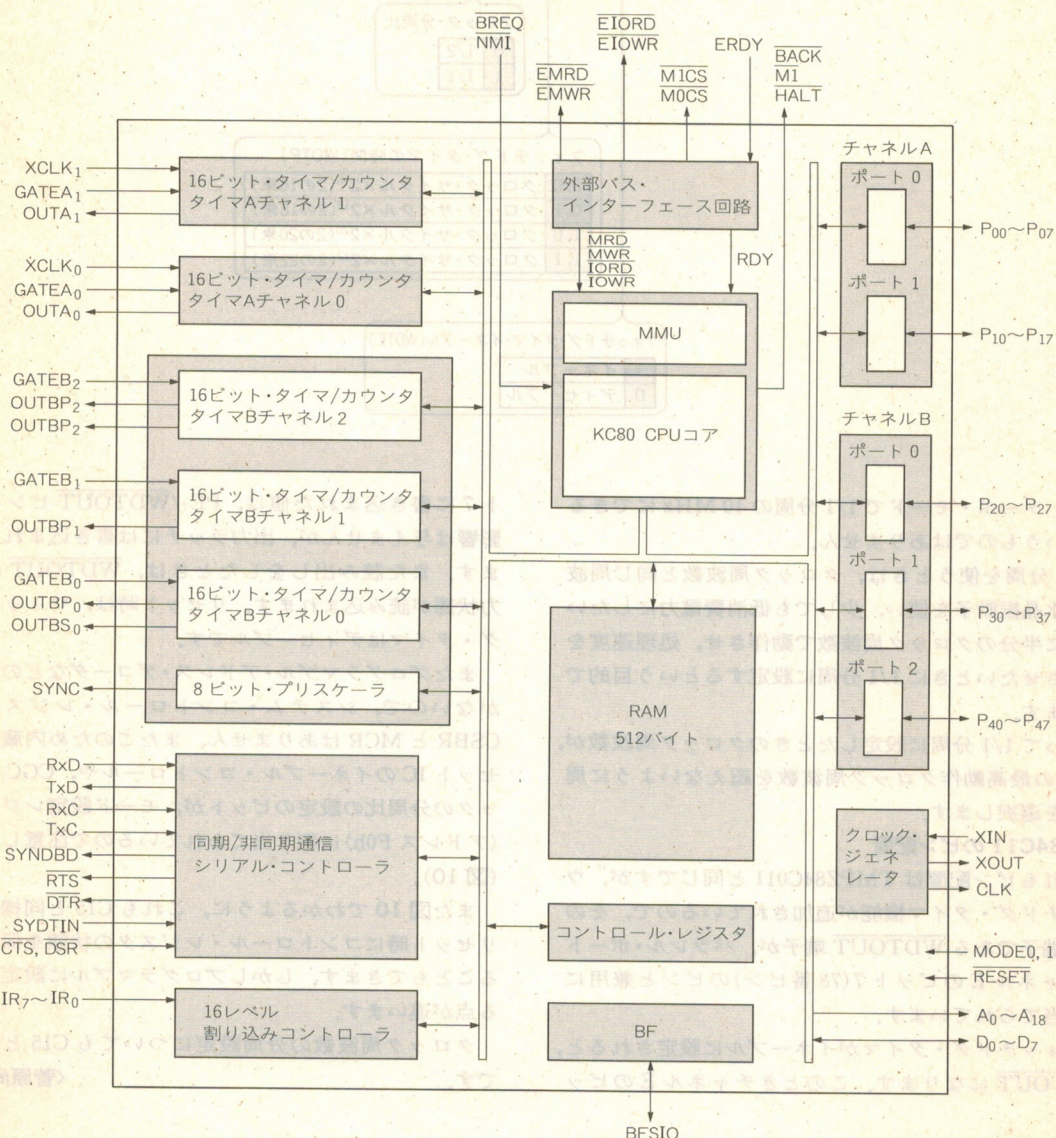
まず**図 11** に内部ブロック図を，また**表 9** に従来の Z80 やその周辺 LSI との違いを示します．

● CPU コア　KC80

KL5C8012 の最大の特徴は，Z80 とソフトウェアがバイナリ・レベルでフル・コンパチブルだという点でしょう．そのため，基本的には従来からのソフトウェア資産をそのまま使うことが可能です．

CPU の最大クロック周波数は 10 MHz です．しかし，一つ一つの命令実行に必要なクロック数が，Z80 と比較して 1/2～1/11 と少なくなっています(**表 10**)．また最小命令実行時間は 1 クロックで 0.1 μs です．

外部メモリや外部 I/O のアクセス・タイミングも高速化されています．ノン・ウェイト時は，1 クロックでリード/ライト・サイクルを実行することになります

〈図 11〉[4] KL5C8012 の内部ブロック

(図12).

Z80は最少命令実行時間が4クロックで，システム・クロック10 MHz時は0.4 μsです.

したがって，同一動作クロック数のZ80と比較しても単純計算で4倍高速となっています.

CPUの信号として外部に出力されているものは，$\overline{M1}$，$\overline{HALT}$，$\overline{NMI}$，$\overline{ERDY}$，$\overline{BREQ}$($\overline{BUSREQ}$)，$\overline{BACK}$($\overline{BUSACK}$)です(括弧内はZ80での名称).

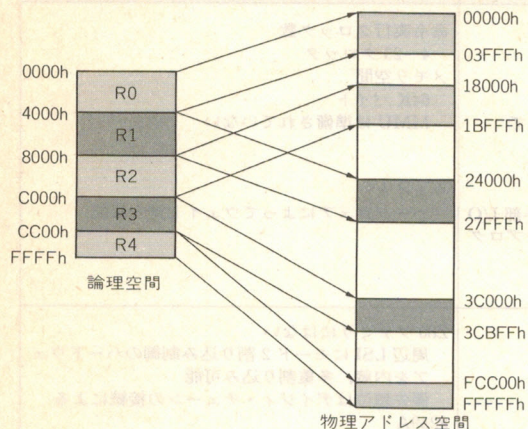
またZ80の$\overline{RD}$，$\overline{WR}$，$\overline{MREQ}$，$\overline{IORQ}$は出力されていませんが，そのかわり，メモリ・アクセス用に

〈表9〉KL5C8012とZ80の比較

	KL5C8012	Z80と周辺LSI
CPU	命令実行クロック数 1～7クロック メモリ空間 一度に扱える範囲は64Kバイト MMUを内蔵し，バンク切り替えによって最大512Kバイトまでアクセス可能 チップ内部に512バイトのRAMを内蔵 ウェイト 外部メモリ・アクセスに1クロック，外部I/Oアクセスに2クロック，ウェイト挿入をプログラムで制御可能 ハードウェアによってさらに延長可能	命令実行クロック数 4～23クロック メモリ空間 64Kバイト MMUは準備されていない ウェイト ハードウェアによってウェイト挿入可能
割り込みコントローラ	16入力(内部8，外部8) モード2対応 エッジ，レベル入力切り替え可 多重割り込み可能 優先順位変更可能	Z80ファミリにはない 周辺LSIにモード2割り込み制御のハードウェアを内蔵，多重割り込み可能 優先順位はデイジィ・チェーンの接続による 8259A モード0使用 8入力 エッジ，レベル入力切り替え可 多重割り込み可能 優先順位変更可能
タイマ/カウンタ	タイマ/カウンタA 16ビット・カウンタ2チャネル カスケード接続により32ビット化可能 4種類の動作モードをもつ タイマ/カウンタB 8ビット・プリスケーラ付き16ビット・カウンタ3チャネル チャネル1はボーレート設定用に使うためシリアル・ポートの送受信クロックに内部で接続可能 4種類の動作モードをもつ	Z80CTC 8ビット・プリスケーラ付き8ビット・カウンタ4チャネル 2種類の動作モードをもつ 8253/8254 16ビット・カウンタ3チャネル 6種類の動作モードをもつ
シリアル・ポート	8251Aと同一機能 1チャネル 非同期式・同期式通信可能	Z80SIO 2チャネル内蔵 非同期式・同期式・SDLC通信可能 CRC生成・チェック機能内蔵 8251A 1チャネル 非同期式・同期式通信可能
パラレル・ポート	パラレル・ポートA 8ビット2チャネル 1ビット単位で入出力の方向制御が可能 出力切り替え時のデータ・プリセット機能 パラレル・ポートB 8ビット3チャネル 4ビット単位で入出力の方向制御が可能 ビット単位のセット/リセットが可能 出力切り替え時のデータ・プリセット機能	Z80PIO 8ビット2チャネル ハンドシェイクによる入出力機能内蔵 ビット・モード時(単純な入出力ポート)，1ビット単位で入出力の方向制御が可能 ビット・モード時，入出力ポートのデータ値により割り込み発生可能 8255A 8ビット3チャネル ストローブ入出力機能内蔵 単純入出力ポート時，ビット単位のセット/リセットが可能

〈表 10〉 実行クロック数の比較

命令の例	KL5C8012	Z80
LD　B, C	1 クロック	4 クロック
ADD　HL, BC	1 クロック	11 クロック
DEC　DE	1 クロック	6 クロック
POP　AF	3 クロック	10 クロック
JR　Z, +20h	3 クロック	12 クロック

〈図 13〉[4] MMU の論理アドレスと物理アドレス

EMRD, EMWR, I/O アクセス用に EIORD, EIOWR が用意されています.

● MMU を内蔵

MMU(メモリ・マネージメント・ユニット)を内蔵することにより, アドレス空間は 1 M バイト(物理アドレス)に拡張されています. ただしアドレス・バスが 19 本しかないので, 外部メモリは最大 512 K バイトまで接続できます(マキシマム・モード時).

また一度に扱えるメモリ容量は Z80 と同じ 64 K バイト(論理アドレス)です. よって 64 K バイトをこえるデータを扱うために, バンク切り替えを行う必要があります.

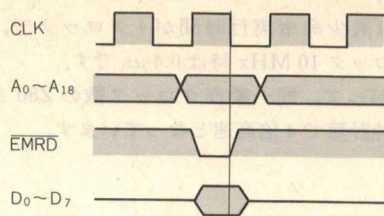
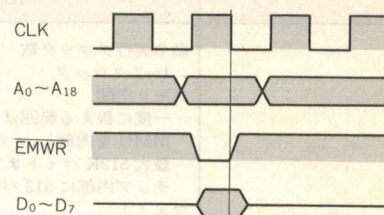
論理アドレス空間は図 13 に示すように五つの領域(R0～R4)に分割されています.

● メモリ・マップ

外部メモリの接続方式はノーマル・モードとマキシマム・モードの 2 種類あり, MODE 端子で設定します.

ノーマル・モードのときは, アドレス・バス A_{18}, A_{17} はチップ・セレクト端子として動作します. したがって A_0～A_{16} までの 128 K バイトのメモリを二つ, 合計 256K バイトまでを制御できます.

マキシマム・モードのときは, 本来のアドレス・バスとして動作するので, A_0～A_{18} までの 512 K バイトとなります.

〈図 12〉[4] KL5C8012 のメモリ・アドレス

(a) 外部メモリ・リード・サイクル

(b) 外部メモリ・ライト・サイクル

また, 内蔵 RAM は物理アドレスで FFE00h ～FFFFFh に配置されています. この RAM は後述のウェイトの設定にかかわらず, 0 ウェイトでアクセスされます.

● ウェイト機能

メモリや I/O の中にはスピードが遅いものもあります. そのためのウェイト制御回路を内蔵しており, 外部メモリ・アクセスに 1 ウェイト, 外部 I/O アクセスに 2 ウェイト入れることができます.

● デバッグ機能

KL5C8012 の特徴として, デバッグ用の専用端子と回路を内蔵している点があげられます.

この専用端子にバグ・ファインダ・アダプタ BFA8001(川崎製鉄)を接続し, パソコンの RS-232-C ポートと接続することによって, ブレーク・ポイントの設定や内部レジスタの読み書き, メモリの読み書きが行えます.

ユーザのメモリや I/O 空間とは異なる空間に置かれるため, ユーザ側の制限はまったくありません. またハードウェア・ブレーク対応など, ROM 上のプログラムにもブレークをかけることができ, ICE 感覚で使用できます.

● スピードの比較

ここでは KL5C8012 の評価ボード SMC01 を使い, Z80 と同一プログラムを実行させたときの処理時間の違いで実行スピードを比較してみます.

比較のための Z80 ボードには TK-Z80(東洋リンクス)を使います. クロック周波数は 4.9152 MHz です. SMC01 の動作クロック数は 10 MHz なので, テスト・プログラムの実行時間を倍にすれば, クロック 5

〈表 11〉実行速度の違い

プログラム番号	SMC01 (10MHz)単位 ms	TK-Z80 (4.9152MHz)単位 ms	同一クロックでのスピード比(TK-Z80を1)
①	39	320	4.03 倍
②	0.1972	1.59	3.96 倍
③	0.162	1.14	3.46 倍
④	0.5943	4.16	3.44 倍
⑤	2.57	17.74	3.39 倍
⑥	10.7	74.1	3.40 倍
⑦	21.9	150.7	3.38 倍

MHz 相当の実行時間になります．

● テスト・プログラムの結果

テストに使用したプログラムは以下のとおりです．

① 単純ループ(65536 回)
② 16 ビット，8 データのソート
③ 8 ビット，256 データからのサーチ
④ 32 ビット×32 ビット
⑤ 64 ビット÷64 ビット
⑥ loge(X)
⑦ デシベル値の計算

表 11 に実測した実行時間と，そのスピード比を示します．

この結果からすると，同一クロック数では 3〜4 倍のスピードアップになっていることがわかります．

①がいちばん速くなりましたが，これは変数をレジスタだけでプログラムしており，ほかはメモリ・アクセスが入っているからだと考えられます．

例えば，LD A，B は Z80 とくらべて 1/4 になるのに対し，LD A，(8000H) は 3/10 となります．

今回のテストでは I/O 入出力を含まないプログラムでしたが，実際にコントローラとして使うには I/O の処理が入ります．

しかし，たとえ外付け I/O を増設し，ウェイトを入れたとしても，I/O 入出力を割り込みで起動させれば，I/O で待たされる間にほかの処理を行うことができるので，全体の処理能力を高めることができるでしょう．

● 使用上の注意点

しかし注意点もあります．まず 100 ピン QFP パッケージにこれだけの機能ピンを装備することは当然無理で，兼用端子が多いため同時に使えない機能があるということです．

また Z80 ベースの CPU とはいえ，バス・サイクルやデイジィ・チェーン接続用端子などの関係で，Z80 ファミリ LSI を直接外付けすることは難しそうです．

〈菊地恭徳〉

KL5C8012の使い方

すでに説明したように，KL5C8012 は多機能な周辺 LSI を内蔵していますが，100 ピン・パッケージに収める関係で，マルチプレクス(兼用ピン化)されているピンがあり，同時に使えない場合があります．**表 12** に内蔵 I/O のアドレスを，**図 14** に兼用ピンになっている入出力ピンの選択レジスタを示します．

また，ここでは KL5C8012 を使った市販マイコン・ボード TC8012(田中電子) と UKC82/01(ユニテク電子)を例にあげて説明します．

● リセット直後の状態

まずリセット直後の状態を知っておかなくてはなりません．注意すべき点は次の 4 点です．

・MMU は一部を除いて外部 ROM を論理アドレス空間に割り付けてしまう
・マルチプレクスされた兼用端子は，すべてパラレル・ポートに選択されている
・ウェイトは外部メモリは 1 ウェイト，外部 I/O は 2 ウェイト
・MODE 端子ピンの状態でノーマル・モード，マキシマム・モード，バグファインダ・モードを決定

したがってリセット後に必要に応じて，内蔵の制御レジスタを設定しなおします．

● MMU の初期化

まず外部 RAM を使用する場合は，MMU の設定をしなくてはなりません．

例にあげた二つのボードでは，MODE 端子がノーマル・モードに設定されているので，ここでは MMU 設定など，ノーマル・モードを例に説明します．

ノーマル・モードでは物理アドレス空間(1M バイト)うちの，00000h 番地から 128K バイト($\overline{\text{M0CS}}$ でアクセス)と，0E0000h 番地から 128K バイト($\overline{\text{M1CS}}$ でアクセス)を，論理アドレス空間の 64K バイトに，五つの領域に分けて割り当てることのできるモードです．

リセット直後は外部 RAM を使うために MMU を設定します．とりあえず Z80 の標準的な使い方と同じように，

・ROM 領域 32K バイト：0000h〜7FFFh
・RAM 領域 32K バイト：8000h〜0FFFFh

と割り付けるためのもっとも簡単な設定を**リスト 4**

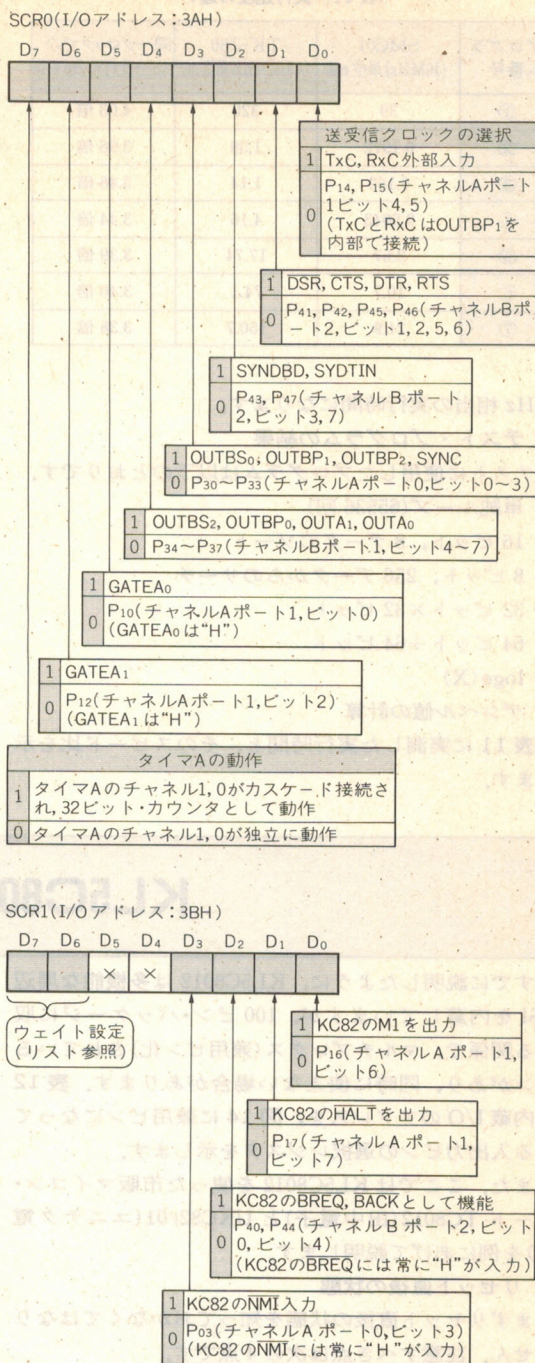
〈表 12〉[4] KL5C8012 内蔵I/Oポート

I/Oアドレス	内蔵I/O	チャネル
00h	KC82 MMU	堺界/ベース・レジスタ1
01h		ベース・レジスタ1
02h		境界/ベース・レジスタ2
03h		ベース・レジスタ2
04h		境界/ベース・レジスタ3
05h		ベース・レジスタ3
06h		境界/ベース・レジスタ4
07h		ベース・レジスタ4
08h-0Fh	川崎製鉄使用予約	
20h	タイマ/カウンタB	チャネル0カウンタ
21h		チャネル0コントロール
22h		チャネル1カウンタ
23h		チャネル1コントロール
24h		チャネル2カウンタ
25h		チャネル2コントロール
26h		川崎製鉄使用予約
27h		川崎製鉄使用予約
28h	タイマ/カウンタA	チャネル0カウンタ
29h		チャネル0コントロール
2Ah		チャネル1カウンタ
2Bh		チャネル1コントロール
2Ch	パラレルA	ポート0データ
2Dh		ポート0方向制御レジスタ
2Eh		ポート1データ
2Fh		ポート1方向制御レジスタ
30h	パラレルB	ポート0データ
31h		ポート1データ
32h		ポート2データ
33h		コントロール
34h	割り込みコントローラ	LERL/PGRL
35h		LERH/PGRH
36h		IMRL
37h		VTAR/IMRH
38h	シリアル・ポート	データ
39h		コマンド/ステータス
3Ah	システム制御レジスタ	SCR0
3Bh		SCR1

〈リスト4〉ROM32K, RAM32KのMMUの設定例

```
;****** MMU設定(ROM32K/RAM32K) ******
        LD      A,1FH
        OUT     (06H),A
```

に示します．MMUレジスタ BBR_4 に1Fhを書き込むだけです．このときは図11でいうところのR0とR4しか有効となりません．

大容量RAMを使えば，余っているR1～R3を使ってバンク切り替えを行うこともできます．ラベルに任意の値を設定すれば，アセンブル時に自動的に計算して MMU を設定してくれるプログラムをリスト5に示すので参考にしてください．

〈図14〉兼用ピン選択レジスタ

● メモリのアクセス・タイムについて

システム・クロックが10 MHzの場合，アクセス・タイム70 nsのROMやRAMが用意できるなら，リスト6の設定でウェイト・コントローラが挿入するウ

〈リスト5〉 MMU割り当て計算プログラム

```
;***********************************************************
;         MMU割り当ての自動計算例                          *
;***********************************************************

;MMUのI／Oアドレス定義
BBR1    EQU     00H     ;MMU REG BBR1
BR1     EQU     01H     ;        BR1
BBR2    EQU     02H     ;        BBR2
BR2     EQU     03H     ;        BR2
BBR3    EQU     04H     ;        BBR3
BR3     EQU     05H     ;        BR3
BBR4    EQU     06H     ;        BBR4
BR4     EQU     07H     ;        BR4

;物理ｱﾄﾞﾚｽ先頭値の定義(R1～R4まで割り当てる物理ｱﾄﾞﾚｽの先頭
;                          を/10Hして定義する)
R1_PS   EQU     00100H  ;各値の下限は00000H,
R2_PS   EQU     0F440H  ;上限は0FFBCH未満
R3_PS   EQU     0F680H  ;ｽﾃｯﾌﾟは40Hとする
R4_PS   EQU     0F800H  ;0F040H以下の値を定義する事

;論理ｱﾄﾞﾚｽ先頭値の定義(R1～R4まで割り当てる論理ｱﾄﾞﾚｽ
;                          の先頭を定義する)
;無効設定をするためには_Bn_に3FHを設定するためには、
;10000Hをここで定義する必要があるが、0FFFFHを越えるﾃﾞｰﾀ
;の定義ができないので、便宜的にRn_LS>=Rn+1_LSと定義する
R1_LS   EQU     01400H  ;ｽﾃｯﾌﾟは400Hとする
R2_LS   EQU     04000H  ;
R3_LS   EQU     08800H  ;
R4_LS   EQU     0C000H  ;

_B1_    EQU     ( ((R1_LS AND 0FC00H) SHR 10) - 1 )
_A1_    EQU     ((R1_PS AND 0FFC0H)-((R1_LS AND 0FC00H)SHR 4))SHR 6
_B2_    EQU     ( ((R2_LS AND 0FC00H) SHR 10) - 1 )
_A2_    EQU     ((R2_PS AND 0FFC0H)-((R2_LS AND 0FC00H)SHR 4))SHR 6
_B3_    EQU     ( ((R3_LS AND 0FC00H) SHR 10) - 1 )
_A3_    EQU     ((R3_PS AND 0FFC0H)-((R3_LS AND 0FC00H)SHR 4))SHR 6
_B4_    EQU     ( ((R4_LS AND 0FC00H) SHR 10) - 1 )
_A4_    EQU     03C0H   ;固定データ

_BBR1_  EQU     ( ( (_A1_ AND 03H) SHL 6 ) OR _B1_ )
_BR1_   EQU     ( (_A1_ AND 03FCH) SHR 2 )
_BBR2_  EQU     ( ( (_A2_ AND 03H) SHL 6 ) OR _B2_ )
_BR2_   EQU     ( (_A2_ AND 03FCH) SHR 2 )
_BBR3_  EQU     ( ( (_A3_ AND 03H) SHL 6 ) OR _B3_ )
_BR3_   EQU     ( (_A3_ AND 03FCH) SHR 2 )
_BBR4_  EQU     ( ( (_A4_ AND 03H) SHL 6 ) OR _B4_ )
_BR4_   EQU     ( (_A4_ AND 03FCH) SHR 2 )

        LD      A,LOW(_BBR1_)   ;MMUﾚｼﾞｽﾀへの書き込み
        OUT     (BBR1),A
        LD      A,LOW(_BR1_)
        OUT     (BR1),A
        LD      A,LOW(_BBR2_)
        OUT     (BBR2),A
        LD      A,LOW(_BR2_)
        OUT     (BR2),A
        LD      A,LOW(_BBR3_)
        OUT     (BBR3),A
        LD      A,LOW(_BR3_)
        OUT     (BR3),A
        LD      A,LOW(_BBR4_)
        OUT     (BBR4),A
;       LD      A,LOW(_BR4_)    ;書き込んでも不変
;       OUT     (BR4),A         ;
```

エイト・サイクルを0にすることもできます．またリセット直後は外部メモリ・アクセスに1ウェイト挿入されていますから，アクセス・タイム150 nsのメモリでも使えます．

USARTの使い方

● 8251互換のSIO

シリアル・コントローラは8251とソフトウェア互換のものが内蔵されているので，8251用のプログラムがそのまま使えます．

またUSARTのTxD，RxD(データ送受信)は専用端子なので，兼用端子の選択設定の必要はありません．

〈リスト6〉 ノン・ウェイト設定プログラム

```
;****** WAITコントローラ初期化 ******
        LD      A,0C0H          ;NO WAIT
        OUT     (3BH),A         ;SET TO SCR1
```

〈リスト7〉 SIO送受信クロックの初期化の例

```
;******* システム・レジスタ初期化 **********
SCR0_INIT:                      ;/DTR,/DSR,/RTS,/CTSを使う
;       LD      A,00000010B     ;5線インターフェースの場合は
;       OUT     (3AH),A         ;ここのコメントを有効にする
;------ タイマ・カウンタB1(クロック・ジェネレータ)初期化
TMRB1_INIT:
        LD      A,00000110B     ;OUTP=0,連続発生ﾓｰﾄﾞ,4分周(GATEなし)
        OUT     (23H),A         ;TCB1Cへのセット
        LD      HL,0007H        ;時間定数のロード
        LD      A,L             ;9.8304MHz/4/(7+1)/2=153.6KHz=9600*16
        OUT     (22H),A         ;下位定数のTCB1へのセット
        LD      A,H
        OUT     (22H),A         ;上位定数のTCB1へのセット
```

またリセット直後はタイマ・カウンタBのチャネル1が，内部でUSARTのTxC，RxC(送受信クロック入力)に接続されているため，そのまま送受信クロックに使用することができます．

リスト7に送受信クロックの初期化プログラムの例を示します．

● カウンタの注意点

タイマ・カウンタの定数は実際のカウント値−1とする点が，Z80CTCとは異なります．また兼用端子には$\overline{RTS}$や$\overline{DTR}$，$\overline{CTS}$がありますが，$\overline{CTS}$については内部でプルダウンされているらしく，3線式のインターフェースでも問題ないようです．

パラレル・ポートの使い方

● パラレル・ポートの設定

汎用パラレル・ポートは，8ビット全5ポートで40ビット分あり，チャネルAは8ビット2ポート，チャネルBは8ビット3ポートです．

チャネルAはビットごとの入出力の指定が可能で，チャネルBは上位下位4ビットごとの入出力や，ビットのセット・リセットが可能です．

またこれらのポートには出力レジスタがあり，入力方向になっていても書き込み値をラッチし，出力方向に切り替えたときにはその値を出力します．

Z80PIOのようなハンドシェイクの制御線はありません．よって入出力時に割り込みを発生させることはできませんが，後述の割り込みコントローラと組み合わせることで同様のことをさせることも可能でしょう．

リセット直後はすべて入力方向に，また内部の出力レジスタも0に初期化されています．図15にチャネ

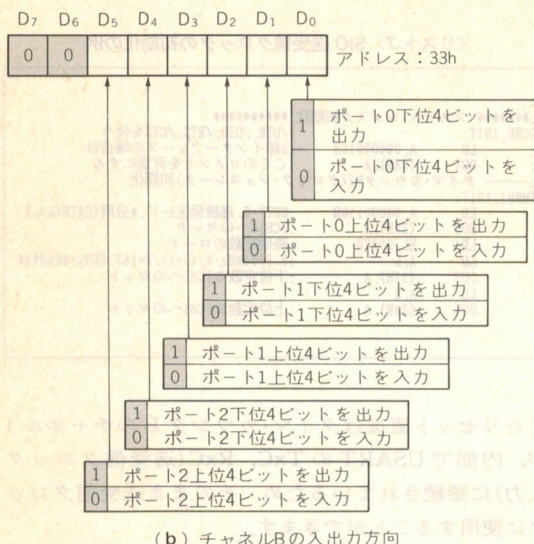
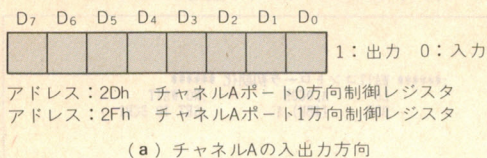
〈図15〉パラレル・ポート方向制御レジスタ

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

1：出力　0：入力

アドレス：2Dh　チャネルAポート0方向制御レジスタ
アドレス：2Fh　チャネルAポート1方向制御レジスタ

（a）チャネルAの入出力方向

（b）チャネルBの入出力方向

ルA，チャネルBの方向制御レジスタの構成を示します．

リセット直後なので兼用ピンの状態はパラレル・ポートが選択されています．よって兼用端子の選択設定なしで，すぐに入力することができます．またチャネルBポート0だけは専用端子となっていて，兼用端子の設定の必要はありません．

● 出力に設定した例

リスト8にチャネルBポート0への出力例を示します．チャネルBの方向はパラレル・ポートB方向制御レジスタ(アドレス33h)の下位6ビットで指定します．

リスト9にチャネルBポート1への出力例を示します．入力状態でデータ・レジスタに書いたデータは方向を出力に指定しても“0”に初期化されるわけでなく，そのまま出力されます．したがって，入力方向で読み出し－>入力方向で書き込み－>出力方向に切り替え－>次のデータの書き込みの手順で，プルアップ/プルダウンにかかわらず，ノイズを出すことなく出力に切り替えることができます．

リスト10にチャネルAポート0のビット0のみの出力例を示します．またリスト11にチャネルBポート0のビットごとの初期化例を示します．ここではビット0のみノイズを出さずに出力に設定した後，ビット操作機能を用いトグルさせています．

〈リスト8〉チャネルBポート0の出力の例

```
;****** ＰＰＢ０出力テスト ************************
        LD      A,00000011B     ;PPB0[7:0]=出力
        OUT     (33H),A
        LD      A,33H           ;出力データ DATA
MAIN:   OUT     (30H),A
        CPL
        JR      MAIN
```

〈リスト9〉チャネルBポート1への出力の例(初期化時にノイズを出さない)

```
;*** ＰＰＢ０出力テスト（ノイズを出さない方法） ***
        IN      A,(31H)         ;PPB1入力
        OUT     (31H),A         ;出力ﾚｼﾞｽﾀへのﾌﾟﾘｾｯﾄ
        LD      A,00001100B     ;PPB1[7:0]=OUTPUT
        OUT     (33H),A

        LD      A,55H           ;OUTPUT DATA
MAIN:   OUT     (31H),A
        CPL
        JR      MAIN
```

〈リスト10〉チャネルAポート0ビット0の出力の例

```
;***** ＰＰＡ０出力テスト ***********************
        LD      A,01H           ;PPA0[0]=出力
        OUT     (2DH),A

        LD      A,01H           ;出力 DATA
MAIN:   OUT     (2CH),A
        XOR     01H
        JR      MAIN
```

〈リスト11〉チャネルBポート0のビット単位の出力の例

```
;*** ＰＰＢ０ビット単位出力テスト（ノイズを出さない方法）
        IN      A,(30H)         ;PPB0入力
        OUT     (30H),A         ;出力ﾚｼﾞｽﾀへのﾌﾟﾘｾｯﾄ
        LD      A,00000001B     ;PPB0[3:0]=出力
        OUT     (33H),A

MAIN:   LD      A,11000001B     ;PPB0[0]=1
        OUT     (33H),A         ;D0 ビット・セット
        LD      A,11000000B     ;PPB0[0]=0
        OUT     (33H),A         ;D0 ビット・リセット
        JR      MAIN
```

タイマ・カウンタの使用例

● タイマ・カウンタAの使い方

タイマ・カウンタAは，

・分周モード
・PWMモード
・パルス・モード
・パルス幅/周期測定モード

の四つの動作モードをもち，外部クロックやシステム・クロックの選択，トリガのエッジなどの組み合わせを含めると，Z80CTCなどとは比較にならないほど高機能な16ビット・タイマ・カウンタです．チャネル0と1の2チャネルを内蔵しています．

〈リスト12〉 タイマ・カウンタAのシステム・クロックの分周の例

```
;***** TMRAチャンネル0 システム・クロック分周テスト *****
        IN      A,(3AH)         ;現在のSCR0設定値の読み込み
        SET     4,A             ;OUTBS2,OUTBP0,OUTA1,OUTA0に
        OUT     (3AH),A         ;97～100ﾋﾟﾝの機能を切り替え

        LD      A,00000001B     ;ｼｽﾃﾑ･ｸﾛｯｸを分周してﾄｸﾞﾙ出力
        OUT     (28H),A         ;ﾀｲﾏ･ｶｳﾝﾀAﾁｬﾈﾙ0ｺﾏﾝﾄﾞ･ﾚｼﾞｽﾀへｾｯﾄ
        LD      HL,0004H        ;分周数=4+1
        LD      A,L             ;下位設定
        OUT     (28H),A
        LD      A,H             ;上位設定
        OUT     (28H),A
```

〈リスト 13〉 タイマ・カウンタ A の PWM モードの例

```
;****** TMRAチャンネル0 PWMテスト **********************
        IN      A,(3AH)         ;現在のSCR0設定値の読み込み
        SET     4,A             ;OUTBS2,OUTBP0,OUTA1,OUTA0に
        OUT     (3AH),A ;97～100ﾋﾟﾝの機能を切り替え

        LD      A,00001111B     ;ﾍﾟﾘｵﾄﾞ=ｼｽﾃﾑ･ｸﾛｯｸの2^12のPWMﾓｰﾄﾞ
        OUT     (28H),A         ;ﾀｲﾏ･ｶｳﾝﾀAﾁｬﾈﾙ0ｺﾏﾝﾄﾞ･ﾚｼﾞｽﾀへｾｯﾄ
        LD      HL,333H         ;ﾊﾟﾙｽ幅=333H+1=820
        LD      A,L             ;下位設定
        OUT     (28H),A
        LD      A,H             ;上位設定
        OUT     (28H),A
```

● 分周器の例

もっとも単純にシステム・クロックを分周する例を**リスト 12** に示します．

最初のシステム制御レジスタ SCR_0への OUT 命令で，端子機能をタイマ・カウンタに切り替えます．ここでは，タイマ・カウンタ A チャネル 0 を使ってシステム・クロックを 5 分周して，$OUTA_0$(OUT 端子)をトグルさせているので，システム・クロックの 1/10 のクロック出力が得られます．16 ビットの定数を 2 回に分けて設定する必要があるので，下位・上位のライト・シーケンスにしたがったコマンド書き込みが必要です．

$GATEA_0$(ゲート入力)は，システム制御レジスタで未使用のままに設定しているので，内部で常に 1 (カウント・イネーブル状態)が入力されています．

● PWM モードの例

同様にシステム・クロックを使った PWM モードの例を**リスト 13** に示します．ここでは，タイマ・カウンタ A チャネル 0 を使ってシステム・クロックの $2^{12}+1=4097$ クロック周期で，333h＋1＝820 クロック時間を H レベルとしました．積分回路(時定数 10 ms 程度)を通すと約 1 V が得られ，簡易的ですが D-A コンバータにもなります．

● パルス幅カウントの例

GATE 端子に入力される信号の，パルス幅を測定する例を**リスト 14** に示します．実験的に，HC4040 を使って $XCLK_0$に 1.2288 MHz，$GATEA_0$にその 1/256 の 4800 Hz を入力してみました．

SCR_0で，$GATEA_0$(入力として使いますが，切り替えが必要)を使用する(イネーブル)に切り替え，測定を立ち上がりでスタートし，立ち下がりで終了するよう設定しました．TCA_0の 1，2 回目は 0FFFFh からダウン・カウントしたカウント値であり，3，4 回目がその補数でカウント値となります．

〈リスト 14〉 タイマ・カウンタ A のパルス幅カウントの例

```
;****** TMRAチャンネル0 :パルス幅測定テスト *************
        IN      A,(SCR0)        ;現在のSCR0設定値の読み込み
        SET     4,A             ;OUTBS2,OUTBP0,OUTA1,OUTA0と
        SET     5,A             ;GATEA0に
        OUT     (SCR0),A        ;79,97～100ﾋﾟﾝの機能を切り替え

        LD      A,00100100B     ;GATEA0の立ち上がりから立ち下がり
                                ;までを、XCLKで1回測定
        OUT     (TCA0C),A

LOOP:   IN      A,(TCA0C)       ;ステータス・リード
        BIT     7,A             ;bit7 チェック
        JR      Z,LOOP          ;測定未終了ならLOOPへ

        LD      A,00111000B     ;カウンタ・ラッチ・コマンド
        OUT     (TCA0C),A

        IN      A,(TCA0)        ;下位ｶｳﾝﾄ値をLレジスタに
        LD      L,A
        IN      A,(TCA0)        ;上位ｶｳﾝﾄ値をHレジスタに
        LD      H,A
        IN      A,(TCA0)        ;下位CRﾚｼﾞをLレジスタに
        LD      L,A
        IN      A,(TCA0)        ;上位CRﾚｼﾞをHレジスタに
        LD      H,A
```

測定結果の HL レジスタの内容は，正しい 0080h (128)の場合と 007Fh(127)の場合がありました．周波数を変えてもみましたが，タイミングによって±1 の誤差は出るようです．

なお，外部カウント基準クロック $XCLK_0$/$XCLK_1$ に関しては入力端子なので兼用端子の切り替えは不要です．しかし端子をポート出力に使った場合は，XCLK を使用するとポート出力がクロックとなるようです．

この他，パルス・モードや 2 チャネルをシリーズにした使い方もできます．

● タイマ・カウンタ B の使い方

タイマ・カウンタ B は，

- PWM モード
- WDT モード
- 連続カウント・モード
- 単発カウント・モード

の四つの動作モードをもつ高機能な 16 ビット・タイマ・カウンタです．チャネル 0～2 の 3 チャネルを内蔵しています．

● 分周器の例

もっとも単純にシステム・クロックを分周する例を，**リスト 15** に示します．動作波形を**図 16** に示します．

タイマ・カウンタ B チャネル 0 を使って，システム・クロックの 4 分周を GATE 機能なしで連続カウントします．ここではカウント初期値を 3 にしたので，4×(3＋1)＝16 分周ごとに $OUTBP_0$端子はトグル動作します．$OUTBS_0$端子には 4 システム・クロックが

〈リスト 15〉 タイマ・カウンタ B の分周器の例

```
;****** ＴＭＲＢチャンネル０ ：連続カウント・テスト ********************
        IN      A,(3AH)         ;現在のSCR0設定値の読み込み
        SET     3,A             ;OUTBS0;OUTBP1,OUTBP2,SYNCと
        SET     4,A             ;OUTBS2,OUTBP0,OUTA1,OUTA0に
        OUT     (3AH),A         ;1,4,5,6,97～100ﾋﾟﾝの機能を切り替え

        LD      A,00000110B     ;4分周ゲート機能なし、連続カウント
        OUT     (21H),A
        IN      A,(21H)         ;ステータス・リード(シーケンス・クリア)
        LD      HL,3            ;ｶｳﾝﾄ初期値=3
                                ;OUTBP0=9.8304MHz/(4*(3+1)*2)=307.2KHz
        LD      A,L             ;下位書き込み
        OUT     (21H),A
        LD      A,H             ;上位書き込み
        OUT     (21H),A
```

〈リスト 16〉 タイマ・カウンタの PWM 出力の例

```
;***** ＴＭＲＢチャンネル０　ＰＷＭテスト ****************************
        IN      A,(3AH)         ;現在のSCR0設定値の読み込み
        SET     3,A             ;OUTBS0,OUTBP1,OUTBP2,SYNCと
        SET     4,A             ;OUTBS2,OUTBP0,OUTA1,OUTA0に
        OUT     (3AH),A         ;1,4,5,6,97～100ﾋﾟﾝの機能を切り替え

        LD      A,00001111B     ;4分周ゲート機能あり、PWMモード
        OUT     (21H),A

        IN      A,(21H)         ;ステータス・リード(シーケンス・クリア)
        LD      A,7             ;パルス周期=7+1
        OUT     (21H),A
        LD      A,2             ;パルス幅=2+1
        OUT     (21H),A
```

〈図 16〉 タイマ・カウンタ B の分周出力波形の例

カウント値 0 のたびに出力されます．ステータスのリードはライト・シーケンス・クリアのために実行します．

● PWM モードの例

リスト 16 に PWM モードの例を示します．動作波形を図 17 に示します．

タイマ・カウンタ B チャネル 0 を使い，システム・クロックを 4 分周した信号を基準にし，$GATEB_0$が H レベルの期間に，8 クロック周期で 3 クロック分の H レベルを出力し，これを繰り返します．これが PWM 出力です．

$GATEB_0$は入力のため端子機能の切り替えは必要ありません．$GATEB_0$=“1” の期間は正しく PWM 波形が出力されますが，$GATEB_0$=“0” の期間ではカウント機能が停止します．図 16 の動作波形では，丁度 2 パルス分に $GATEB_0$期間がありますが，$GATEB_0$が $OUTBP_0$=“1” の期間中に “0” になったりすると，波形だけでは正しい PWM といえなくなります．このほかにも，単発カウント・モードや，WDT モード，システム・クロックの 256 分周までのプリスケーラの設定などあります．

割り込みコントローラの使用例

● Z80 モード 2 対応

割り込みコントローラは，Z80 のモード 2 割り込みをサポートしています．割り込み要因としては，

- 内蔵タイマ・カウンタからの 5 要因
- 内蔵 USART からの 3 要因
- 外部割り込みリクエスト（IR_7～IR_0）ピンからの 8 要因（IR_7～IR_0は，パラレルポート A チャネル 0 と兼用）

の 16 個がサポートされています．割り込みの優先度や許可，ベクタなどを制御するレジスタが 6 個あります．

高機能なので比較的簡単な外部割り込みリクエスト・ピンからの割り込み例をリスト 17 に示し，簡単な解説をします．

● 外部割り込み

リスト 17 では，最初に各種レジスタの設定を行います．レベル/エッジ・レジスタ，ベクタ・レジスタ，マスク・レジスタの設定を行っています．この例では割り込み番号 IR_3～IR_0を許可としました．

プログラム例は実験的に外部から 40 k，20 k，10 k，5 kHz の順に，IR_0，IR_1，IR_2，IR_3へ HC4040 の出力を与え，エッジ・モード（立ち上がり）で割り込みを受けた回数をワーク・エリアに書き込みます．IR_3受け付け終了時に割り込みディセーブル状態でメイン・ルーチンに戻り値を確認します．

グループ・レジスタを設定していないので，割り込み優先度はデフォルトの IR_3～IR_0の順になり，値はそれぞれ 4，2，1，1 となり，正しい値となりました．

多重割り込みの優先度に関しては Z80 ファミリの IEI，IEO のデイジィ・チェーン結線や信号のディレイなどを気にすることなくプログラムで設定できます．

このほか，タイマ・カウンタや USART からの割り込み，IR_7～IR_0端子をポート出力として切り替える，

〈図 17〉 タイマ・カウンタ B の PWM 出力波形の例

〈リスト17〉割り込みプログラムの例

```
;****** 割り込みコントローラテスト ＩＲピン多重割り込み ***********
        LD      SP,9000H
        DI
        IM      2               ;ﾓｰﾄﾞ2使用
;       LERH/L初期化、リセット後はレベル・モード
        LD      A,0FH           ;IR[7:4]レベル,IR[3:0]エッジ
        OUT     (34H),A
        LD      A,0             ;IR[15:8]レベル
        OUT     (35H),A
;       IVR初期化
        LD      HL,INTVEC
        LD      A,H
        LD      I,A             ;上位ベクタをIレジスタにセット
        LD      A,L
        OUT     (37H),A         ;下位ベクタをIVRにセット
;       IMR初期化
        LD      A,0F0H          ;IR[7:4]不可,IR[3:0]許可
        OUT     (36H),A
        LD      A,0FFH          ;IR[15:8]不可
        OUT     (37H),A
CHECK:  IN      A,(2CH)         ;IR3(PPA0[3])が
        BIT     3,A             ;1なら待つ
        JR      NZ,CHECK
        EI                      ;割り込み可
LOOP:   JR      $               ;無限ループ

INTIR0: EI                      ;EI命令を実行しないと
        LD      A,(IR0_CNT)     ;多重割り込みできない
        INC     A
        LD      (IR0_CNT),A
        EI
        RETI

INTIR1: EI
        PUSH    AF
        LD      A,(IR1_CNT)
        INC     A
        LD      (IR1_CNT),A
        POP     AF
        EI
        RETI

INTIR2: EI
        PUSH    AF
        LD      A,(IR2_CNT)
        INC     A
        LD      (IR2_CNT),A
        POP     AF
        EI
        RETI

INTIR3:                         ;最高優先度のためEI不要
        PUSH    AF
        LD      A,(IR3_CNT)
        INC     A
        LD      (IR3_CNT),A
        POP     AF
;       EI                      ;リターンするだけで次の割り込みは
        RETI                    ;受け付けない

        ORG     ($+1FH) AND 0FFE0H
INTVEC: DW      INTIR0          ;IR0
        DW      INTIR1          ;IR1
        DW      INTIR2          ;IR2
        DW      INTIR3          ;IR3
        DW      0               ;IR4
        DW      0               ;IR5
        DW      0               ;IR6
        DW      0               ;IR7
        DW      0               ;IR8
        DW      0               ;IR9
        DW      0               ;IR10
        DW      0               ;IR11
        DW      0               ;IR12
        DW      0               ;IR13
        DW      0               ;IR14
        DW      0               ;IR15

        ORG     8000H           ;RAM領域
IR0_CNT:DB      0
IR1_CNT:DB      0
IR2_CNT:DB      0
IR3_CNT:DB      0
```

ソフトウェア割り込みなどができます．

● 最後に

最後にリスト18にKL5C8012の内蔵I/Oポートのアドレスを定義したファイルを示します．ヘッダ・ファイルとして活用してください．

ここまで高機能のCPUなら，TMPZ84C015のようなHALTモードでの低消費電力化や，外部I/Oをアクセスするためのプログラマブル・チップ・セレクト端子などが内蔵されると，さらに使いやすくなるような気がします．

〈北　孝〉

〈リスト18〉KL5C8012の内蔵I/Oアドレスのヘッダ・ファイルの例

```
;************************************************************
;   KL5C8012 : INTERNAL I/O REGISTER ADDRESS DEFINITION
;************************************************************
BBR1    EQU     00H     ;mmu Border/Base Reg 1
BR1     EQU     01H     ;               Base Reg 1
BBR2    EQU     02H     ;        Border/Base Reg 2
BR2     EQU     03H     ;               Base Reg 2
BBR3    EQU     04H     ;        Border/Base Reg 2
BR3     EQU     05H     ;               Base Reg 3
BBR4    EQU     06H     ;        Border/Base Reg 4
BR4     EQU     07H     ;               Base Reg 4

TCB0    EQU     20H     ;Timer/Counter B-ch0 count reg
TCB0C   EQU     21H     ;                 // Control reg
TCB1    EQU     22H     ;Timer/Counter B-ch1 count reg
TCB1C   EQU     23H     ;                 // Control reg
TCB2    EQU     24H     ;Timer/Counter B-ch2 count reg
TCB2C   EQU     25H     ;                 // Control reg

TCA0    EQU     28H     ;Timer/Counter A-ch0 count reg
TCA0C   EQU     29H     ;                 // Control reg
TCA1    EQU     2AH     ;Timer/Counter A-ch1 count reg
TCA1C   EQU     2BH     ;                 // Control reg

PPA0    EQU     2CH     ;Parallel Port A-ch0 data reg
PPA0D   EQU     2DH     ;                 // Direction reg
PPA1    EQU     2EH     ;         Port A-ch1 data reg
PPA1D   EQU     2FH     ;                 // Direction reg

PPB0    EQU     30H     ;Parallel Port B-ch0 data reg
PPB1    EQU     31H     ;         Port B-ch1 data reg
PPB2    EQU     32H     ;         Port B-ch2 data reg
PPBD    EQU     33H     ;Parallel Port B-Direction reg

LERL    EQU     34H     ;Level/Edge      Reg-Low  for write
LERH    EQU     35H     ;Level/Edge      Reg-High for write
PGRL    EQU     34H     ;Priority Group  Reg-Low  for write
PGRH    EQU     35H     ;Priority Group  Reg-High for write
ISRL    EQU     34H     ;In Service      Reg-Low  for read
ISRH    EQU     35H     ;In Service      Reg-High for read
IMRL    EQU     36H     ;Interupt Mask   Reg-Low  for write/read
IMRH    EQU     37H     ;Interupt Mask   Reg-High for write/read
IVR     EQU     37H     ;Interupt Vector Reg      for write

SIO     EQU     38H     ;Sirial I/O data reg
SIOC    EQU     39H     ;Sirial I/O Control reg

SCR0    EQU     3AH     ;System Control Reg0
SCR1    EQU     3BH     ;                // Reg1
```

◆参考・引用*文献◆

(1) 8ビットマイクロプロセッサ TLCS-Z80ASSP, 1991年, ㈱東芝.

(2) VOLUME1 DATABOOK, MICROPROCESSORS AND PERIPHERALS, ZILOG.

(3) Z80 Family User's Manual, ZILOG.

(4)*高速8ビットマイクロコントローラKL5C8012ハードウェアマニュアル，川崎製鉄㈱.

(5) 高速8ビットマイクロコントローラKL5C8012アプリケーションノート，川崎製鉄㈱.

(トランジスタ技術1994年6月号および10月号に加筆，修正)

第11章

アセンブラによるソフト開発からICEによるデバッグまで

Z80マイコン・システム開発手順

野口智樹/藤丘勝信

Z80 マイコン・システムの開発手順

マイコンが理解できる言葉 機械語

● 機械語とは何か

皆さんご存じのとおり，マイコン(CPU)はプログラムがないとまったく動作してくれません．マイコンは一つ一つの命令を解読し，それを忠実に実行するだけです．

さて，そのマイコンが理解してくれる命令とはどんな命令なのでしょう．もうすでにおわかりと思いますが，マイコンが理解できる命令とは，人間が使っている言葉のようなものではなく，01hや3Ehといったような，数値で表されたものです．

このような，マイコンが直接に理解できる数値による命令が機械語と呼ばれるものです．

命令といっても人間から見れば単なる数字の羅列です．人間が機械語を読んでみてもすぐに理解できるものではありませんし，またその必要もほとんどありません．人間がマイコンのプログラムを作成するために，もっと簡単に理解できるように，機械語と1対1に対応させて表現した言語がアセンブリ言語です(図1)．

● アセンブラとは

大むかし(といっても十数年前)までは，アセンブリ言語から機械語への変換作業は人間が1行ずつ手作業で行っていました．現在ではこのような変換はソフトウェアが自動的に行ってくれます．このようなソフトウェアのことをアセンブラといいます．

また今のアセンブラでは，単なる機械語への変換だけではなく，プログラム開発が簡単になるようにいろいろと便利な機能が備わっています．これらの機能についてはここでは省きます．

● アセンブリ言語のメリット

世の中にはさまざまなプログラム言語がありますが，その中でもっともCPUに近い部分のプログラミングはアセンブリ言語によって行われています．

CPUに近い部分というのは漠然とした言い方かもしれませんが，アセンブリ言語は機械語と1対1で対応している言語ですから，CPUの動作の一挙手一投足までをコントロールすることができるということです(図2)．

つまりアセンブリ言語を使えば，CPUのもつパフォーマンスを100%引き出すことができます．

● アセンブリ言語のデメリット

アセンブリ言語には当然デメリットもあります．高級言語(C言語やFORTRAN，BASICなど)に比べ，一般的にソフトウェアが大規模になるほど開発効率が低下するなどといった問題点もあります(アセンブラのほうが高級言語より得意だ！ という人もいるかも

機械語	マイコンの処理	アセンブリ言語
3E 01	Aレジスタに1を代入	LD A, 1
21 34 12	HLレジスタに1234hを代入	LD HL, 1234h
3C	Aレジスタに1を加算	INC A
FE 02	Aレジスタと2を比較	CP 2
CA 78 56	ゼロ・フラグが1ならば5678hへの分岐	JP Z, 5678h

↑ 数字の羅列では人間はわからない．

↑ これなら人間にわかりやすいがいちいちこんな文章で書くのはめんどう．

↑ 人間に意味がわかる程度にまで記号化してマイコンのプログラムを書く．

〈図1〉 機械語とアセンブリ言語の対応

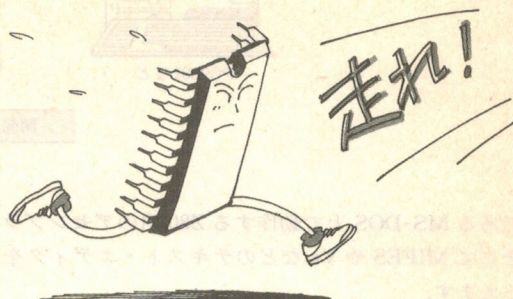
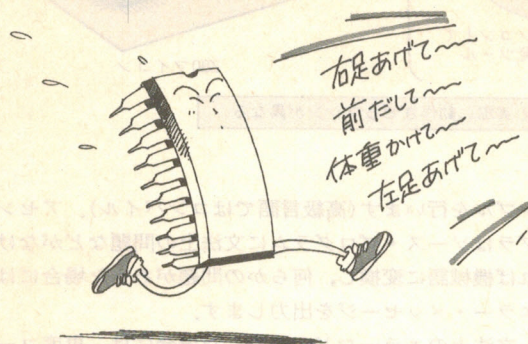
〈図 2〉アセンブリ言語は CPU を細くコントロールできる

LD　A，12H	A レジスタに 12h を代入
CP　B	B レジスタと比較
JP　Z, JMP1	ゼロ・フラグが 1 なら JMP1 へジャンプ

CPU 動作の一つ一つを細かく指定できる

〈図 3〉高級言語は一つの命令でアセンブリ言語の数十行分にあたる

```
if　(A>B)　printf(“Hello！”)：
　else printf(“CQ！”)：
```

一つの命令でアセンブリ言語の数十行にあたる

しれませんが…）．

なぜアセンブリ言語では開発効率が低下するのでしょうか．例えば小数点演算などがある場合，アセンブラでは数十行ものプログラムになりますが，高級言語では 1 行で済んでしまうからです．

このようにアセンブリ言語では，その一つ一つの命令が非常に簡単なことしかできないので，何か意味のある動作をさせるためにはいくつもの命令を組み合わせなければなりません．ところが高級言語は 1 行の命令だけでも意味のある動作をさせることが可能なのです．このことから高級言語に対する言葉として，アセンブリ言語のことを低級言語と呼ぶこともあります（図 3）．

以上のことから，プログラムの処理速度やメモリ容量にかなりの余裕があるならば，ほとんどの部分に高級言語を使い，高速な処理が必要な部分だけをアセンブリ言語でプログラミングすれば，バランスの良いシステムが作れるのです．

マイコン・システムの開発手順

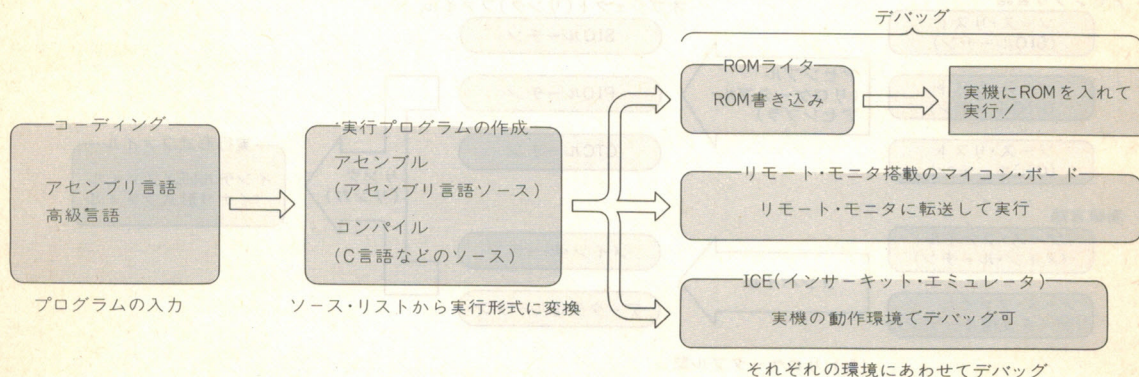
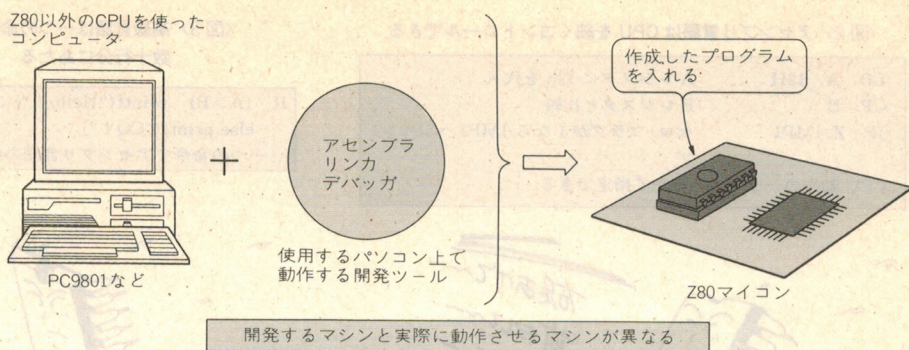
Z80 は一般に小型の組み込み用機器などで多く使用されているマイコンですが，このような組み込み機器を目的とする場合のソフトウェア開発は，どのような手順で行えばよいのでしょうか．

基本的な開発手順は図 4 のようになります．

● 開発環境の整備

マイコンの開発環境としては，現在では一般にパソコンが使われているようです．この場合，当然アセンブラなどは開発用のパソコン上で動作するものが必要となります．また，アセンブラ以外にもコーディングを行うためのエディタ，ROM 書き込みのためのソフトウェアや書き込み器なども必要になります．

例えばパソコンとして PC-9801 を使う場合は，OS

〈図 4〉プログラムの開発手順

〈図5〉
クロス開発とは

である MS-DOS 上で動作する Z80 用のアセンブラ，そして MIFES や Vz などのテキスト・エディタをそろえます．

また，PC-9801 で使われている CPU は Z80 とはまったく別の CPU です．このように，ある CPU のアセンブル作業を別の CPU で動作するプログラムによって行うことをクロス開発などと呼び(図5)，このとき使用するアセンブラを，クロス・アセンブラと呼びます．

● **コーディング**

コーディングとはソース・プログラムを作る作業のことです．つまり LD A, (1234H) というようにアセンブリ言語を記述していく作業です．

プログラムを入力するためにはワープロなども使えますが，一般にはさきほど説明したようなエディタと呼ばれるソフトウェアを使います．

● **アセンブル**

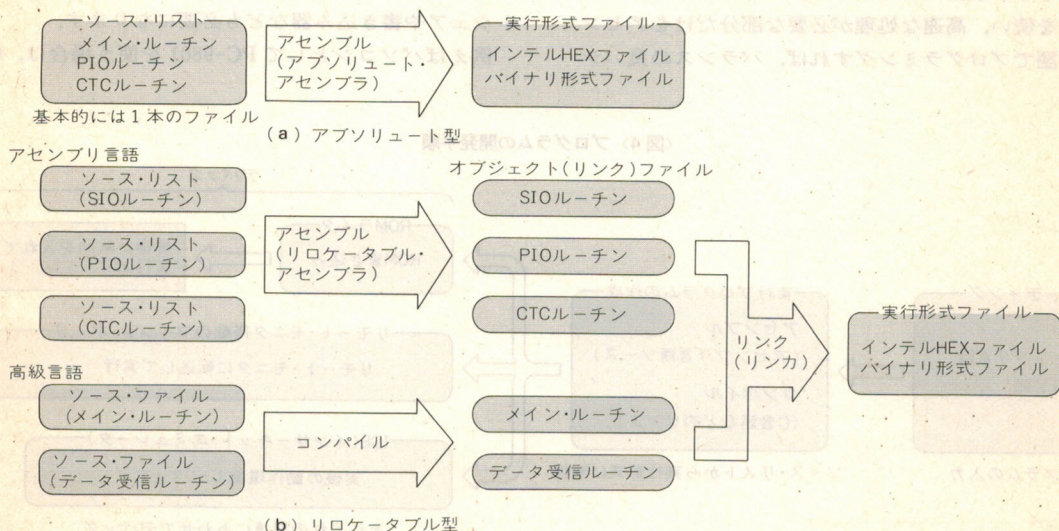
コーディングが終わると，アセンブリ言語で書かれたソース・プログラムを機械語に変換するためにアセンブルを行います(高級言語ではコンパイル)．アセンブラはソース・プログラムに文法上の問題などがなければ機械語に変換し，何らかの問題があった場合には，エラー・メッセージを出力します．

文法上のエラーなどが発生した場合には，再度コーディングに戻り，エディタで間違いを訂正してアセンブルします．

またアセンブラには，アブソリュート型とリロケータブル型とがあり，場合によって使い分けします(図6)．

アブソリュート型とは，ソース・リストから直接実行形式ファイルを出力するタイプのアセンブラです．開発するプログラムの規模が小さい場合はこれで十分でしょう．

リロケータブル型とは大規模プログラム開発に適しています．ソース・リストを機能ごとに分割し，いくつかのファイルからオブジェクト・ファイルと呼ばれる中間形式のデータを作成します．そして必要なオブジェクト・ファイルを，リンカと呼ばれるツールで結

〈図6〉 **アブソリュート型とリロケータブル型の違い**

合し，実際に動作できる実行形式ファイルを作成するものです．またC言語などの高級言語と合わせてプログラムを開発するには，リロケータブル型アセンブラを使います．

● デバッグ

作成した実行プログラムを実際に動作させて，不具合がないかどうかチェックする行程です．

この行程は使用するツールによって作業の流れがいろいろあります．いちばん基本的なやり方は，ROMライタでプログラムを書き込み，実機のZ80マイコンのROMソケットに挿入して動作させる方法です．

二つめはZ80マイコン・ボードにリモート・モニタを搭載して，パソコンと通信しながらデバッグする手法です．

三つめはICE(インサーキット・エミュレータ)を使う方法で，レジスタの値の変化やプログラムのステップ・実行などが可能です．

● プログラムが動かないわけ

さて，プログラムが正常に動かないのならなぜアセンブラがエラーを出さなかったのでしょうか？

アセンブラとは，文法や命令のスペル・ミス，未定義の名前や範囲外の値が代入されたときなどしかエラーを出力しません．

つまり，プログラムの論理的なミスは人工知能でも搭載しないかぎりチェックは不可能なのです．

また，パソコンでのプログラミングとマイコンでのプログラミングの大きな違いとして，パソコンでは(本来なら)ハードウェアは完全に動作することがわかっていますが，マイコンでは，プログラムの間違いによるものではなく，ハードウェアの間違いの可能性も考えられる点です．

どちらにしろ，プログラムが動かなければ，どこに原因があるかを突き止めるデバッグという作業に入ります．

● デバッガとは

ROMに書かれた機械語プログラムは，電源投入直後から一気に実行されるので，動かない原因を探るために人間が「あ，いまAレジスタに10hが代入されたなぁ」などとやってる暇はありません．

そこで，デバッグ作業を効率よく行うために，一般的にはデバッガというツールを使います．

デバッガにはいろいろな種類がありますが，デバッグ手法については後でまとめて解説します．

アセンブラによるソフトウェア開発

すでに説明したように，マイコンのソフトウェア開発にはアセンブラや高級言語が使われます．ここでは，いちばん基本的なアセンブラによるソフトウェア開発について解説します．

Z80アセンブラの基礎

● アセンブリ言語の構文

アセンブリ言語には文法や記述における作法，そして必ずそうしなければならないと決まっているわけではないが，慣習的にそうしているといったことなどがあります．

図7にアセンブリ言語の構文を示します．アセンブリ言語はラベル，ニモニック，オペランド，そしてコメントで1行が構成されています．

〈図7〉アセンブラ言語の構文

▶ ラベル(Label)

今までの例ではアドレスを指定するのに直接1234hなどとアドレス値を指定しましたが，機械語が数字の羅列でわかりづらいのと同様，そのアドレスは何のデータを保存しておくアドレスなのか直感的にわかりません．

例えば，Aレジスタに外部スイッチの状態が入っていて，これを保存しておくメモリであれば，

```
LD (8005H),A
```

と書くよりは，

```
LD (SWSAVE),A
```

と書いたほうが，スイッチの状態をセーブしておくアドレスなんだということがわかります．

このSWSAVEがラベルであり，プログラムの各所で参照される場合に命名する，アドレスの名前のことです．

ラベルは一般的には第1カラム(1行の最左端)から書くことになっていますが，多くのアセンブラでは必ずしもその必要はありません．ただし，ニモニックよりも左側に書かなければなりません．

通常，ラベルの最後には“：”(コロン)を付加しま

す．ラベルとして使用可能な文字は，英数字(A～Z, a～z, 0～9)と“_”に“@”などですが，先頭の文字だけは数字であってはなりません(ラベルか値かの判別がつかなくなる場合がある)．

ニモニックを書き始める行まではTABキーで移動するのが一般的なので，“:”とニモニックの間に1文字空白を入れると，ラベルに使える文字の長さは6文字となります．このことからラベルの文字数は伝統的には6文字以内で書くことになっていますが，現在では，多くのアセンブラが10文字以上の長いラベル名でも認識するようになっています．

長さにとらわれず，直感的にわかりやすいラベル名をつけたほうがよいでしょう．

▶ ニモニック(Mnemonic)

ニモニックとはアセンブラの命令のことで，Z80の場合LD A,Bの例でいえばLDの部分のことです．

▶ オペランド(Operand)

オペランドとはニモニックなどの命令に続くパラメータのことです．この書式はニモニックによって決まります．

▶ コメント(Comment)

1行の中で“;”(セミコロン)があると，それより右側の文字列はすべてコメントとみなされます．

〈表1〉数値定数の表現

表現	基数	例	
～B	2進数	110B	10111010b
～O	8進数	17O	12o
～Q	8進数	17Q	17q
～D	10進数	100D	100d
～	10進数	100	−123
～H	16進数	1AH	0A4h

コメント部分の使い方は自由です．細かいプログラムの説明や解説などを記入しておくのがよいでしょう．

● 数値定数の表現

数値定数は一般的に表1のように表現します．2進数は“B”を，16進数には“H”をつけます．また16進数で最上位桁がA～Fで始まるときはラベルと区別するために頭に“0”をつけます(例：FFFFHのときは0FFFFH)．

またアセンブラによっては10進数で“D”を，8進数で“O”や“Q”が使えるアセンブラもあるようです．

● 数式を使った値の代入

一般的なアセンブラでは，値の代入に数式も使えます．

例えば，3バイトである意味をもつデータが5組あ

コメント記述はマクロな目で

よくプログラムの1行ごとに詳細なコメントを記述しているリストを見受けます．それでプログラムが本当に読みやすいのであれば問題ないのですが，かえってゴチャゴチャとわかりにくくなる場合もあります．

また例えば**リストA**のようなコメントは，ニモニックの動作を日本語にしただけのことでほとんど意味がありません(初心者のために，Z80の動作を示す意味なら別ですが)．

よほど複雑なプログラムでない限りは，マクロ(大まか)な見かたでコメントを記述したほうが読みやすい場合が多いと思います．

そのサブルーチンで処理している内容や，例えばCP Bというような，AレジスタとBレジスタの比較も，何と何のデータを比較しているのかなどをコメントに書くべきです．

また実際に仕事でマイコンのプログラムを作る場合，プログラム・リスト以外に，ハードウェアや，通信データのフォーマットなどの細かな仕様書を用意します．

よって，何のデータか忘れてしまいそうな文字列

〈リストA〉意味のないコメント，意味のあるコメント

```
LD      A,B         ;B.regのデータをA.regに代入
ADD     A,C         ;A.regとC.regを足す
```

(a)日本語アセンブラ(?)の意味しかないコメント

```
LD      A,1011011B    ;bit7 割り込み可  bit6 カウンタMode
OUT     (CTC0),A      ;bit2 タイム・コンスタント書き込み

DEFB    "ADC",13,10   ;A-D変換開始コマンド
```

(b)ハードウェアの仕様書がなくてもわかるコメントの例

データの意味や，ビットごとに処理が割り当てられたポートの設定部分には，簡単なフォーマットやポートのビットの意味をコメントとして記述し，だいたいの意味はリストを見ただけでわかるようになると，見やすいリストであるといえるでしょう．

もっとも，コメントがなくてもスラスラと読めるようなプログラムが書ければ，バグも少なくなって最高なんですが…

〈リスト1〉
HALTプログラム
のソース・リスト

```
CSEG……………プログラム領域であることを示す
ORG      0………アドレス0000h番地から次の命令やデータを格納する
HALT ……………Z80CPUのための命令
END ……………ソース・リストの終了を示す
```

〈表2〉アセンブラの疑似命令

疑似命令	書式		機能	使用例
ORG		ORG式	ロケーション・カウンタの設定	ORG 100H
EQU	ラベル	EQU式	ラベルの値設定	DATA EQU 80H
DEFB		DEFB式,…	バイト型データの定義	DEFB 45H,67H
DEFW		DEFW式,…	ワード型データの定義	DEFW 9000H
DEFS		DEFS式,…	領域定義	DEFS 1024

IF
ELSE
ENDIF
		IF式
		ELSE
		ENDIF
			条件アセンブル
			式が真ならELSEまたはENDIFまでを,偽ならELSE以降をアセンブル
				IF INPUT EQ 1
				OUT (PIOA), A
				ELSE
				IN A, (PIOA)
				ENDIF

CSEG		CSEG	プログラム・コード領域の定義	CSEG
DSEG		DSEG	データ領域の定義	DSEG
INCLUDE	INCLUDE"ファイル名"		ファイルの読み込み	INCLUDE"IOADR.H"
END		END	プログラムの終了指定	END

ったとき，そのデータ・ブロックの全バイト数を単純に，

```
LD    A,15
```

と指定しても，プログラムに間違いはありません．しかしこれを，

```
LD    A,3＊5
```

と記述することによって，3バイトのデータが5個で15という値が出てきたことを表現できます．

またラベルやシンボルに対しても数式が使えます．例えば，DATAというシンボルを10hとしているとき，Aレジスタにこの半分の8を入れるとします．このとき，

```
LD    A,8
```

ではなく，

```
LD    A,DATA/2
```

とすることもできます．このように記述することで，単にAレジスタに8を代入しているということだけでなく，DATAというシンボルの半分の値を代入しているのだという意図を表現することができます．

使用できる数式はアセンブラによってさまざまで，単純な四則演算しかできないものから，シフト命令や論理演算までできるものもあります．

アセンブラ疑似命令の使い方

● 疑似命令とは

リスト1にもっともプログラム・サイズの小さいHALTプログラムを示します．HALT命令とは，CPUの動作を停止する命令です．CPUの外部から見れば，Z80CPUのHALT出力端子をLレベルにするだけのものです．

実行したい本来の内容はHALT命令の1命令だけですが，ソース・リストにはCSEGやORG，ENDなどの文字列がついています．Z80の命令書をみても，ORならありますがORGなどという命令はないはずです．

このプログラムの4行のうち，実際のZ80CPUへの命令はHALTの1行だけです．残りの3行はアセンブラに対しての指示であり，これらは機械語にはなりません．

このような命令は疑似命令とよばれ，機械語は生成されませんがその意味はとても重要なのです．いくらZ80の命令を理解していても，このような疑似命令を正しく使わないと思いどおりのプログラムはできません．

ここでは疑似命令を活用した実践的なアセンブラ・プログラムの組み方を説明します．まず基本的な疑似命令の効果的な使い方を解説します．

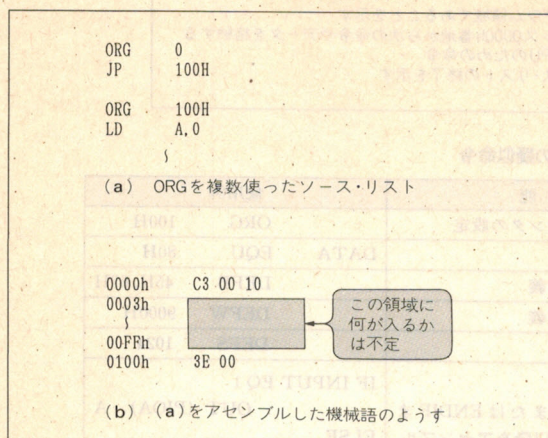
表2に疑似命令の例を示します．

これから解説する疑似命令は，使い方次第で開発効率や保守性を向上させることができます．

とくにINCLUDEやIF～ELSE～ENDIFなどの疑似命令は，上級者ほどこれらの疑似命令を上手く使い，プログラムをさまざまな仕様にカスタマイズできるような構造にしたり，デバッグの効率を高めたりしています．

〈リスト 2〉 ORG の使用例

〈リスト 3〉 EQU の使用例

〈リスト 4〉 コメントの使用例

● ORG

これはアセンブラが生成する機械語やデータを格納するアドレスを指定するものです．厳密にはロケーション・カウンタを設定する疑似命令です．ロケーション・カウンタとは，アセンブラが内部で管理している，現在アセンブル中のアドレスのことです．

Z80 のプログラムならば，リセット後 0000h 番地から実行を開始するので，0000h 番地から命令を記述します．

またこの疑似命令は，**リスト 2** の例のように一つのプログラムの中で何度でも使うことができます．

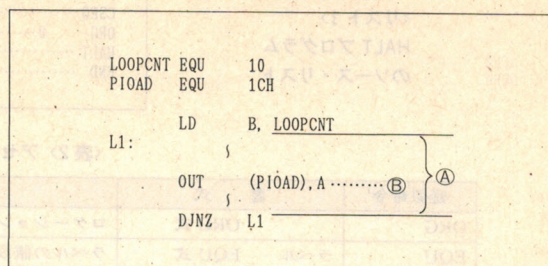
さてこの**リスト 2** の場合，JP 命令は 0000h 番地から格納され，3 バイト命令です．ソース・リストでは次の行にまた ORG 命令が入り，100H 番地となっています．よってその次の行の LD A, 0 は 0100h 番地から格納されます．

では，この 0003h～00FFh 番地の間はどうなるのでしょう．実はこの領域は通常は不定になります．直接バイナリ・データを出力するアセンブラでは，00h か FFh を埋めるものが多いようです．

もっともこのような未使用領域は，どんな値が入っていようとプログラムの動作に影響はないはずなので，通常はとくに考える必要はないでしょう．

● EQU

シンボル(定数)定義命令で，おそらくもっとも多く使用される疑似命令でしょう．**リスト 3** に使用例を示します．この例では，A の部分の処理を LOOPCNT 回，つまり 10 回繰り返します．また B は I/O アドレス PIOAD，つまり 1Ch に A レジスタのデータを出力することを意味しています．

直接 OUT (1CH), A と記述しても動作はまったく同じです．しかしシンボルで記述することにより，そのアドレスがパラレル・ポートのチャネル A のデータ・ポートであるということを明示できるわけです．

またハードウェアの変更で，I/O アドレスが移動することも考えられます．もし直接アドレスの値を記述していた場合，ソース・リストのいたるところに散らばっている OUT (1CH), A という部分のアドレスをすべて書き換える必要があります．

これがシンボルで定義してあれば，ソース・リスト先頭の EQU 文を書き換えるだけですみます．

しかし，シンボルを乱用しすぎるとプログラム・リストを眺めていてもラベルの値を覚えていなければかえって読みにくくなることもあります．

● コメント(；)

アセンブラは，コメント(；)より右側の文字を認識しません．したがってシンボルやラベル，ニモニック以外の文字列を記述することができます(**リスト 4**)．ここにはプログラムの説明を書き込むことができます．

● DEFB, DEFW

DEFB はバイト型(1 バイト)の，DEFW はワード型(Z80 の場合 2 バイト)の定数データを定義します．使用例を**リスト 5** に示します．

また DEFB は文字列の定義も，シングル・クォーツ(あるいはダブル・クォーツ)により区切って表現できます．さらに複数のデータを並べたり文字列の後ろにバイナリ・データを入れたり，シンボルやラベル(アドレスの名前)を記述することもできます．

またアセンブラによっては，文字列の定義に DEFB は使えず，文字列格納専用の疑似命令として，DEFM や DEFC を使う場合もあります．

さらに省略形として DB, DW が使えるアセンブラもあります．

またデータが実際にメモリに格納される順番にも気をつけます．ワード・データは上位下位が逆になって格納されます．

〈リスト5〉DEFB, DEFWの使用例

```
DEFB   10 ………………… 1バイトの10(16進数で0Ah)
DEFW   10 ………………… 1ワードの10(16進数で000Ah)

DEFB   'HELLO'………… 5バイトデータ
DEFB   "HELLO"………… 5バイトデータ

DEFB   10, 20 ………… 二つ以上の定義はカンマで区切る
DEFB   'Hello', 0 …… 文字列と数値を1行に書ける
DEFW   MAIN ………… ラベルの値も入る
```

〈リスト6〉DEFSの使用例

```
        ORG     8000H   ;RAM領域

BUFF:   DEFS    10……10バイトのメモリを確保
                    メモリ領域の先頭アドレスは
                    ラベルBUFF
```

〈リスト7〉INCLUDEの使用例

```
INCLUDE "Z84C015.DAT"
        ∫
        OUT     (PIOAD),A
        OUT     (PIOBD),A
        ∫
```

(a) TEST.ASMの例

```
PIOAD   EQU     1CH
PIOBD   EQU     1EH
```

(b) Z84C015.DATの内容

このように，プログラム中で使用するパラメータや定数，変数の初期データを定義するものです．よってマイコンのプログラムでは，この疑似命令はROM領域で使用します．RAMは電源OFFでその内容は消えてしまいます．したがってRAM領域に初期データを定義することはできません．

しかし変数は，プログラム実行中に値を変更するわけですから，値を書き換えられるRAM上に割り当てられなければなりません．これには次のDEFSを使います．

● DEFS

DEFSは，指定したバイト数のメモリ領域を確保します．例えば**リスト6**のようにすると，10バイトのデータ領域が確保され，その先頭アドレスはラベル名BUFFという名前になります．

これも省略形としてDSが使えるアセンブラもあります．

これは先述のDEFBやDEFWと違い，特定のデータを初期データとしてもたせることはできません．確保したメモリ領域の先頭アドレスと，それ以降に指定されたバイト数分のメモリ領域が確保されたにすぎません．

この領域をとくに初期化する必要がある場合は，その領域を使用する前に，プログラムで初期化する必要があります．

DEFSは，文字列の受信バッファとして使ったり，読み込みデータの一時退避バッファとして使うのが一般的なので，一般的にはRAM領域に定義して使います．

● INCLUDE

INCLUDEは，指定されたファイルをそのソース・リストに取り込む命令です．アセンブラはINCLUDE文があった場合，そのソース・リストの位置に指定されたファイルを挿入し，一つのソース・ファイルであるとみなしてアセンブルします．

このINCLUDE疑似命令は，つぎのような場合によく使用されます．

- いくつものプログラム・ファイルがあり，それらに共通で利用されるEQU定義などをINCLUDEで取り込む
- とくに汎用的なプログラム・ファイル(ライブラリ)を取り込む
- 仕様の異なる部分だけをINCLUDEで取り込み，メイン部分は共通化する

リスト7に使用例を示します．TEST.ASMにはシンボルPIOADを定義している部分はありません．しかしZ84C015.DATをINCLUDEすることで，その部分にリスト(b)の文字列が埋め込まれたソース・リストであるとみなされるので，エラーは出ないで正常にアセンブルされます．

● IF～ELSE～ENDIF

条件付きアセンブル命令で，条件式が成立したときはIF～ELSEの間をアセンブルします．成立しないときはELSE～ENDIFの間をアセンブルします．

リスト8にIF～ELSE～ENDIFの使用例を示します．このリストでは動作させるCPUがTMPZ84C015の場合は，シンボルCPUの値を015に，TMPZ84C011のときは011にします．

Z84C015のパラレル・ポートの初期設定方法と，Z84C011のパラレル・ポートの初期設定方法が異なるため，シンボルCPUの値によりCPUを判別して条件付きアセンブルをしている例です．

データを出力する方法はそのままOUT命令を実行すればよいので，ENDIFの後に記述します．ただしそのI/OアドレスはCPUによって異なります．このI/Oアドレスの定義も，条件付きアセンブルの中で定義されているので，どちらのCPUでも動作するソー

```
        CPU     EQU     015
                 ⁝
        IF CPU EQ 015
        PIOAD   EQU     1CH             ;PIO Ach データ・ポート
        PIOAC   EQU     1DH             ;PIO Ach コントロール・ポート
                LD      A,11001111B     ;モード3 ビット・モード
                OUT     (PIOAC),A
                LD      A,11111111B     ;全ビット出力
                OUT     (PIOAC),A
                LD      A,00000111B     ;割り込み使用しない
        ELSE
        PIOAD   EQU     50H             ;PIO Ach データ・ポート
        PIOAC   EQU     54H             ;PIO Ach コントロール・ポート
                LD      A,11111111B     ;全ビット出力
                OUT     (PIOAC),A
        ENDIF
                LD      A,B
                OUT     (PIOAD),A       ;データ出力
                 ⁝
```


〈リスト8〉
IF〜ELSE〜ENDIFの使用例

ス・リストを記述できます．

ただしこの命令も，乱用すると見通しの悪い，読みにくいプログラムになります．

● CSEG，DSEG

CSEGはコード・セグメント，DSEGはデータ・セグメントの宣言文です．コード・セグメントとはROM（プログラムや定数データ）領域，データ・セグメントはRAM（変数データなど）領域と考えてください．

これらの疑似命令はリロケータブル型のアセンブラに用意されているもので，効率的なモジュール化プログラミングを支援するものです．

CSEG/DSEGを各モジュールごとに宣言し，プログラムとデータ領域を明確に記述することで，最終的なアドレス割り当てはプログラムを連結するリンカが決定し，プログラム領域やデータ領域を特に意識する必要はなくなります．

これらの疑似命令を使わないと，一つのメイン・モジュールですべての変数領域の割り当てを管理しなければならなくなります．これではモジュール化する意味が損なわれてしまいます．

アブソリュート型アセンブラでは，とくに意味はありません．逆にアセンブラによっては対応していない命令としてエラーを出す場合もあります．

● 算術演算子

+，−，/，*などは，アセンブラがもっている算術演算子です．まちがってもZ80にこれらの算術演

C言語とアセンブラの上手な使い分け

筆者は5〜6年前から，Z80でもCコンパイラを中心として必要な部分だけアセンブラでROM化する手法を用いています．

また，人間がアセンブラで作成したコードと，同様の処理をCで記述しコンパイルしたコードとを比較した場合，やはりコンパイラの出力するコードはまだまだ高速化の余地があります．

またサイズ的にも，コンパクトになるようなコード生成を意識しないでプログラムした場合は2〜3割，意識しても1割程度，プログラム・サイズが大きくなります．

しかし，このようなデメリットよりも，開発とデバックの効率が飛躍的に向上するメリットのほうが大きいので，ほとんどCで書くようにしています．

特にサブルーチンをライブラリ化するとき，アセンブラの場合にはどうしてもレジスタがこれこれで…などといったことを考えなければなりませんし，ソース・プログラムの行数も多くなって読みにくくなってしまいます．

Cならばサブルーチンなど完全にブラック・ボックス化できるので，よけいなことに気をつかう必要はありません．

また，より高速化するためにはアセンブラで書いたモジュールをCのサブルーチン（関数）にすることで，問題はほとんどなくなります．

Cのソースレベル・デバッガは，本格的に効率良く開発を進めたい方には必須でしょう．筆者はiD-1600A（コンピュテックス）というROMエミュレータ型のデバッガを使用しています．通常は単なるROMエミュレータとして使っているだけですが，原因が突き止めにくいバグにはソース・レベルのステップ実行でテストしています．

〈表3〉
演算子のいろいろ

算術演算子	( ) + − / * MOD(剰余) SHR(右シフト) SHL(左シフト) AND OR XOR
論理演算子	EQ(等しい) NE(等しくない) GE(以上) LT(より小さい) GT(より大きい) LE(以下) NOT(ビット反転)
その他	HIGH(16ビットの値の上位8ビットを取り出す) LOW(16ビットの値の下位8ビットを取り出す)

注)論理演算子はおもにIF文で使用される．偽が“0”，真が“1”．

(a) 各種演算子の例

高位	( )
↑	単項の+，−
優	* / MOD SHL SHR
先	2項の+，−
順	LT LE EQ GE GT NE
位	NOT
↓	AND
低位	OR XOR
	HIGH LOW

注)優先順位が同位の場合は出てきた順番に処理される．

(b) 算術演算子の優先順位

〈リスト9〉算術演算子の使用例

```
DATA    EQU     16

        LD      A,DATA*2
        OUT     (PIOAD),A
```

(a)演算子を使ったリスト

```
        LD      A,32
        OUT     (PIOAD),A
```

(b) (a)と同じ動作のリスト

〈リスト10〉ロケーション・カウンタ($)の使用例

```
        JP      $       ;('JR  $'も同じ)
```

(a) 無限ループの例

```
LOOP:   JP      LOOP    ;('JR LOOP'も同じ)
```

(b) ラベルを使った無限ループの例

算子が使えるわけではありません．一般的なアセンブラでは**表3**のような演算子が使用できます．

ほかにもいろいろな演算子が用意されているアセンブラもあります．また，ANDが .AND. や&などと，表現が若干異なる場合もあります．

ところで，アセンブラがもっている演算子とはどういう意味でしょう．**リスト9**をみてください．**リスト9**(a)も**リスト9**(b)も，アセンブラから出力される機械語データは同じ，つまり**リスト9**(a)の場合は計算結果が格納されるわけです．このようにアセンブラは，アセンブルと同時に計算もしてくれるわけです．

またすでに**リスト8**で示しているように，条件付きアセンブルの条件式にも使われます．

● ロケーション・カウンタ($)

アセンブラが内部でもっている，現在アセンブル中のアドレスを示す変数で，命令コードやデータを格納するたびに，そのバイト数だけインクリメントされます．またORG命令による指定は，この変数に値を設定することと考えることもできます．

よく使われる例として**リスト10**のような，無限ループにする例があげられます．説明するまでもないでしょうが，これは**リスト10**(b)とまったく同様な動作をします．

● END

アセンブリ・ソース・ファイルが終わったことを宣言します．アセンブラはEND文をみつけると，それ以降の文を無視してアセンブル作業を終わります．

よってこれ以降のソース・ファイルの内容については，いっさい関知しません．END文以降の行は，コメントと同じように使用することができます．

なおアセンブラによっては，END文がなくてもファイルの最終に到達した時点で，アセンブル作業を終了するものもありますが，このようなアセンブラであってもEND文は入れるべきです．〈野口智樹〉

(トランジスタ技術1994年6月号および10月号より)

MS-DOS用アブソリュート・クロス・アセンブラの使い方

アセンブラにはアブソリュート型とリロケータブル型があることはすでに説明しました．リロケータブル型は大規模プログラムの開発に最適で，さらにマクロ命令と呼ばれる機能により命令を拡張できるなど，アセンブラでありながら高級言語並みに使えます．

ただしそのぶん機能がありすぎて，初心者の方には使いづらい面があるのも事実です．ここではいちばん基本的な機能をおさえた，アブソリュート型アセンブラ ZAA.EXE の使い方(入手方法については後述)を説明します．

● 特徴と動作環境，仕様

ZAA.EXE はインテル HEX 形式の実行形式ファイルやリスティング・ファイルを出力する，日本語 MS-DOS 用の Z80 アブソリュート・クロス・アセンブラです．動作環境や仕様を**表 4** に，実行画面(ヘルプ・メッセージ表示時)のようすを**図 8** に示します．

日本語 MS-DOS 用ですから，エラー・メッセージやヘルプ・メッセージなどが日本語で表示されます．

アブソリュート型としてひととおりの機能は備わっています．前述の疑似命令もすべて使えます．算術演算子も使えます．

名前やラベルに使える文字は，A〜Z，0〜9，"_"，"."，漢字も使えます．ただし，先頭の文字に 0〜9 の数字は使えません．また現在のロケーション・カウンタを示す値として$も使えます．

● アセンブル作業の流れ

まず普段使い慣れているテキスト・エディタでソース・リストを入力します．ファイルの拡張子は .ASM にします．

アセンブルは，DOS のプロンプトから次のように入力します．

　　>ZAA　ファイル名　/H /S /L[リターン]

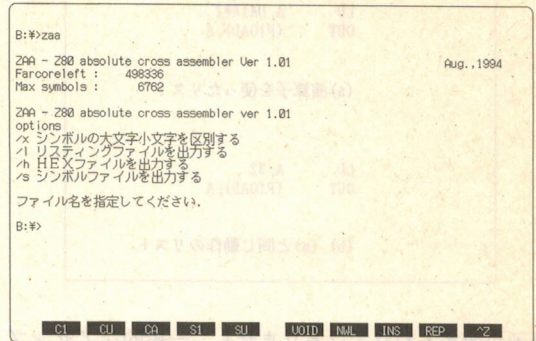
拡張子の .ASM は省略できます．ここで例えば Z80TEST.ASM というソース・リストをアセンブルすると，Z80TEST.HEX という HEX ファイルと，

〈リスト 11〉Z80.BAT の内容

```
ECHO OFF
REM
REM   ZAA用 アセンブル・バッチ・ファイル
REM
IF "%1" == "" GOTO HELP
ZAA %1 /H /S /L
GOTO QUIT

:HELP
ECHO アセンブルするファイル名を指定してください。

:QUIT
```

〈表 4〉ZAA.EXE の動作環境および仕様や制限

動作環境	OS	日本語 MS-DOS Ver.3 以上
	メモリ	フリーエリア 384 K バイト以上
仕様や制限	・一行の長さは MAX130 文字程度 ・シンボルは，長さ無制限(実際は数十文字程度) ・シンボル・ファイルで出力できるのは 44 文字まで ・ニーモニックはザイログ形式 ・未定義命令(LD IXH，n)はサポートしない ・アセンブルした結果が 64 K を超えるようなことがあっても無視する ・シンボル数の最大値は，起動時のメモリの残りによって決定される ・EQU の前方参照は不可	

〈図 8〉ZAA の起動画面

Z80TEST.LST というリスティング・ファイル，Z80TEST.SYM というシンボル・ファイルを出力します．

インテル HEX とは，ROM ライタなどにプログラムを書き込むときなどによく使われるファイル形式で，拡張子が .HEX となります．後述する ROM エミュレータ型 ICE でも使用します．

リスティング・ファイルとは，ソース・リストにアセンブルした機械語やアドレス情報を付加したリストです．シンボル・ファイルとは，そのプログラムの中で使用されたラベルや名前などを示すファイルです．

なお，これらのファイル形式などの中身については特に知らなくてもとりあえずはかまいません．

アセンブルするたびにこのようなオプション・スイッチを付けるのも面倒なので，**リスト 11** のようなバッチ・ファイルを作って使いやすくします．使い方は，

　　>Z80　ファイル名[リターン]

とするだけです．

● エラーへの対処

アセンブル時に表示されるエラーメッセージは，シンボルやラベルが定義されていないときや，文法上のエラーなどがあったときに表示されます．エラーが発生すると何行目が何のエラーかを表示するので，先ほどソース・プログラムを入力したときと同様に，テキ

スト・エディタでソース・プログラムを呼び出して，エラーのあった行を修正します．

図9にエラーが発生したときの画面表示のようすを示します．

● エラー対処例

ここではいくつか代表的なエラーと，その修正方法や対策方法を説明します．

▶ アセンブルできないコード

Z80では使えない命令を指定したときなどに表示されます．

```
        ADD             C,B
```

CレジスタとBレジスタの直接の足し算はできません．Z80は命令の直交性が悪いので，ありそうでない命令を使ってしまうことがよくあります．命令表でよく確認してください．

▶ 0で除算した

その名のとおりゼロで割り算したときに表示されます．もっとも普通は，

```
        LD              A,5/0
```

などと直接ゼロを書くことはないと思われます．

```
        LD              A,10/DATA
```

などのとき，DATAの値がゼロのときに表示されます．

▶ 定数が不正

文字列を切り出して数値に変換するとき，文字が残った場合などに表示されます．

```
        LD              HL,1A6
```

文字の最後に "H" がないと16進数とはみなされません．または途中の "A" が余計です．

```
        LD              A,0002000B
```

2進数表記なのに "2" が入っています．

▶ 値がない

本来なら値や文字がくるべき所になにもないと表示されます．

```
        LD              (IX+),1
        DEFB            1,2,
```

▶ 括弧などの対応がおかしい

括弧やダブルコーテーションの囲みの対応がおかしいときに表示されます．

```
        DEFB            "ABVC
        INCLUDE         "IOADR.H
```

最後にダブルコーテーションが足りません．

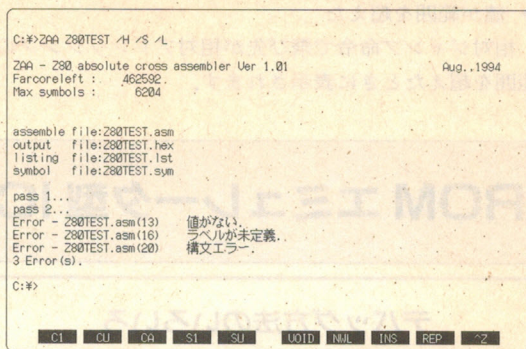
```
        DEFB            'DKJLFKAJDL"
```

文字列の囲みはシングルコーテーションでもダブルコーテーションでもできますが，どちらかに統一してください．

```
        LD              A,(DATA1+(DATA2+1)
```

計算式の括弧が足りません．

```
        LD              (HL+1),A
```

〈図9〉 ZAAのエラー・メッセージのようす

HLレジスタではディスプレイスメント付きのインデックス・アドレッシングはできません．IXやIYレジスタを使えば大丈夫です．

▶ 余分な文字列が付いている

命令や構文上，文字のいらないところに文字がきていると表示されます．

```
        LD              A,B  :comment
```

コメントの指定に ":" を間違って使った例です．

```
        LD              A,(DATA+2))
```

括弧が多いときも表示されます．

▶ ラベルまたは名前が不正

ラベルには予約語や先頭が数字の文字列は使えません．

```
        LD              A,LD+4
```

▶ ラベルまたは名前の二重定義

同じ名前のラベルや名前があるときに表示されます．

```
name    EQU             100H
name:   LD              A,H
```

▶ ラベルが未定義

定義されていないラベルを使うと表示されます．

```
        LD              A,hjk
```

ラベルの綴り間違いということが多いので，スペルを確認してください．また，

```
        LD              A.B
```

のように "," と間違って "." を使ってもこのエラーが表示されます．

▶ 構文エラー

ラベル，名前，予約語以外の文字で行が始まるときなどに表示されます．

```
100:    LD              A,B
```

先頭の文字が数字の場合はラベルになりません．

```
LD:     XOR             A
```

予約語はラベルに使えません．

```
        (IX+4),A
```

いきなりオペランドがきている例です．

```
        LD              B,
```

","の次がありません．

▶ 値が範囲を超えた

相対ジャンプ命令で飛び先が相対アドレッシングの範囲を超えたときに表示されます．

```
        JR          Z,JP_ADR1
```

JP命令にするか，飛び先のアドレスを近い位置に変更します．

ROMエミュレータ型ICEを使ったデバッグのすすめ

デバッグ方法のいろいろ

● デバッガとは

すでに説明したように，アセンブラの表示するエラー・メッセージとは，文法上のエラーについてだけで，プログラムの論理的な間違いについては関知していません．

よってCPUがどのようにプログラムを実行していくか，レジスタやメモリはどう変化していくかを見ることができると，プログラムの動作そのものわかり，プログラムの動作のチェックが非常にしやすくなります．

このように，プログラムを実行したときのレジスタの変化のようすや，メモリの内容を表示したり編集できたるするツールを，デバッガとよびます．

● ROMライタだけ！

デバッグ・ツールが何もない状況では，できたプログラムをROMに書き込んで動作するかどうかを確認する方法しかありません．動作しなければまたプログラムを直して再アセンブルし，ROMの書き直しが必要です．

また，パソコンの画面やプリンタに打ち出したリストを眺めるという机上デバッグしか手がないので，タイミングに問題のある場合などは，その原因を突き止めるのが困難になります．

● ターゲット側にモニタを乗せる

次に考えられる手法としては，動作させるマイコン側に，何らかのモニタ・プログラムを搭載し，このモニタ・プログラムを使いながらレジスタやメモリの変化を確認しプログラムの動作を調べるものです．

スタンド・アロンで動作させたいなら，最低限16進キーによる入力や7セグメントLEDによる表示が必要です．しかしデバッグが終われば不要になります．組み込み機器としてはデバッグ時だけのハードウェアは無駄になります．このスタイルのマイコン・ボードは基本的に学習用ということになるでしょう［図10(a)］．

そこで，入力や表示は他の装置にまかせ，ターゲット側にはモニタ・プログラムとその装置との通信部分のハードウェアを搭載させる形のモニタが考えられました．これがリモート・モニタとよばれるものです［図10(b)］．

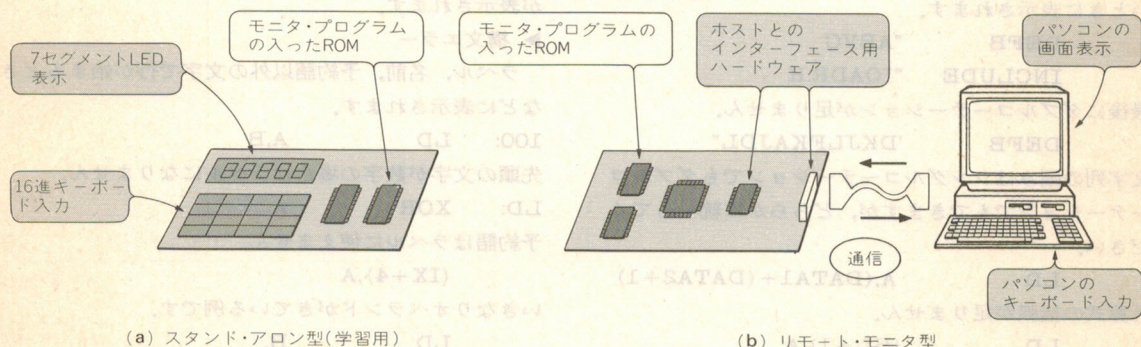
モニタ・プログラムの機能としては，レジスタやメモリの表示や書き換えの機能は同じですが，その結果をシリアル・ポートなどを通じてパソコンなどとやり取りする部分がスタンド・アロン型と異なります．

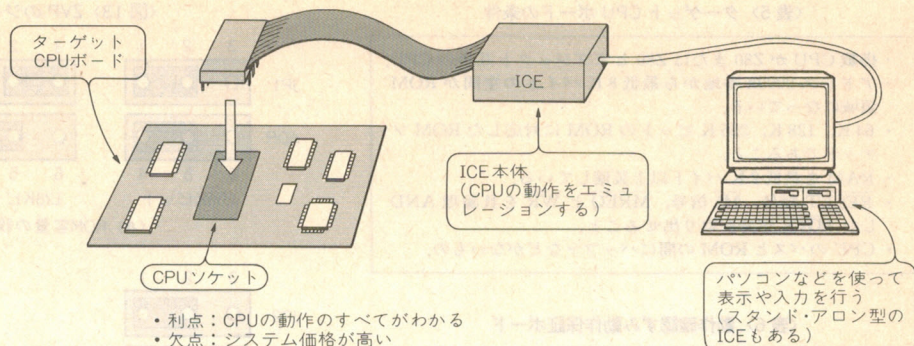
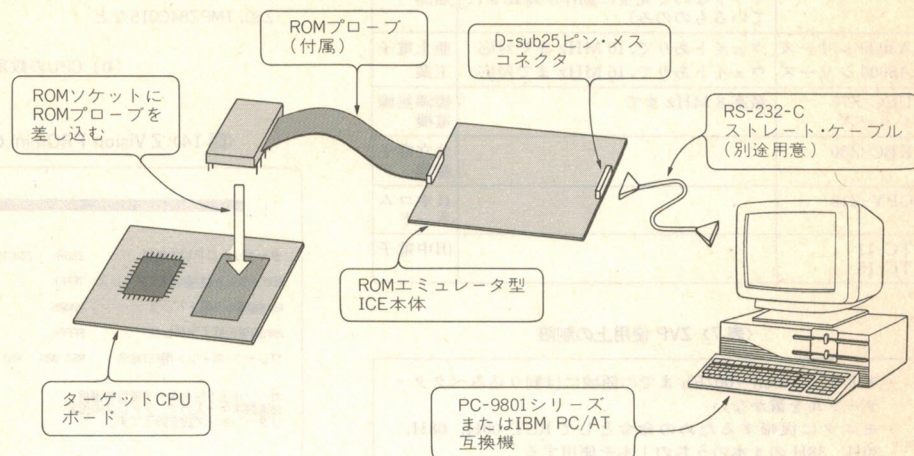
● リモート・モニタを使ったデバッグ

ROMライタだけの環境よりは格段に"まともなデバッグ"ができる環境にはなりました．しかし，ホスト側からデバッグしたプログラムを転送して実行したり，メモリを書き換えたりしてデバッグを進めるわけですから，RAM上で作業しなければなりません．ROM領域でのメモリの書き換えやステップ実行などが不可能です．

またマイコンの限られたメモリ空間に，モニタ・プ

〈図10〉モニタ・プログラムの形態

(a) スタンド・アロン型(学習用)　(b) リモート・モニタ型

〈図11〉
ICEによるデバッグ

〈図12〉
ZVPのシステム構成

ログラムも入るわけですから，開発するプログラムが大規模になるとメモリが足りなくなることもあります．

● ICEによるデバッグ

ICE(イン・サーキット・エミュレータ)とはCPUそのものをエミュレーションさせることで，CPUの動作をその手の内までも調べられる，マイコン開発ツールとしては最高のデバッグ・ツールです(図11)．

CPUに電源やクロックが正常に入力されているかどうかの物理的なチェックから，ROM領域のメモリの読み書き，ステップ実行なども可能になります．また基本的にマイコンのメモリ空間にモニタ・プログラムのための領域やワークエリアが取られることがありません．

しかしやはりネックはその価格にあるでしょう．Z80用といえど，多機能なものは数十万円はします．また使用するCPUのパッケージがQFPだったりすると，取り付けるためのアダプタが必要になったり，取り付け不能な場合すらあります．

● ROMエミュレータ型ICEによるデバッグ

ここでは，ICEと同等の機能をもちながら，それでいてリモート・モニタ並の価格と使いやすさの，ROMエミュレータ型ICE Z Vision PRO(システムロード)を活用したデバッグ手法について簡単に解説します．

Z Vision PROmini

● ZVPの概要

Z Vision PRO(以下ZVP)とは，ROMプローブをターゲットのROMソケットに，RS-232-C端子をホスト・パソコンに接続して使用する，ROMエミュレータ型ICEです(図12)．

ここではZVPの機能限定版であるZ Vision PROminiを使用してみます．mini版といっても，ひととおりのデバッグ機能は備わっています．

その主な特徴は，

▶ ROM領域へのダウンロード，ブレーク・ポイントの設定，ステップ実行が可能

▶ Z80のもつメモリやI/Oの全アドレス空間をユーザに解放

▶ ROMソケットに接続するので，取り付け方法がCPUのパッケージに左右されない

などが上げられます．

対応するターゲットCPUボードの条件を表5に，

〈表5〉ターゲットCPUボードの条件

- 搭載CPUがZ80またはZ80とオブジェクト互換のCPU.
- アドレス0000h番地から最低8Kバイトの空間がROM領域になっている.
- 64K, 128K, 256KビットのROMに対応したROMソケットであること.
- RAMを最低2Kバイト以上装備している.
- $\overline{RESET}$信号, $\overline{M1}$信号, $\overline{MREQ}$と$\overline{WR}$を負論理ANDした$\overline{MWR}$信号が取り出せること.
- CPUのバスとROMの間にバッファなどがないもの.

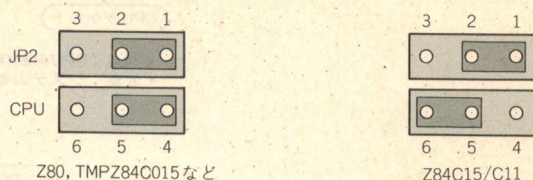
〈表6〉動作確認ずみ動作保証ボード

ボード型名	最高クロック周波数	メーカ
AKI80	最高クロック周波数8MHz(ただし,キットなので完全に動作が確認されているもののみ)	秋月電子通商
A3000シリーズ A5000シリーズ	ウェイトありで,10MHzまで対応 ウェイトありで,16MHzまで対応	亜土電子工業
UEC-Z77シリーズ	最高8MHzまで	梅澤無線電機
KBC-Z50		共立電子産業
GPY-Z15		日本コムネット
TC-11 TC-15		田中電子

〈表7〉ZVP使用上の制限

- アドレス0000h~007Fhまでの領域には割り込みベクタ・テーブルを置かない
- モニタに復帰するための命令としてRST 00H, 08H, 30H, 38Hの4本のうちの1本を使用する
- CPUの最高クロック周波数は8MHzまで(ウェイト挿入ができれば16MHzまで)

〈図13〉ZVPのジャンパ設定

(a) ROM容量の設定(JP1)

(b) CPUの設定(JP2)

〈図14〉Z Vision PROminiのセットアップ画面

動作確認済みのマイコン・ボードを**表6**に示します.

またメモリやI/Oの制限は基本的にはありませんが,**表7**に示すような若干の制限があります.

● 前準備

まず使用するターゲット・ボードに合わせて各モード・スイッチをセットします.まずターゲットROMソケットに合わせてROM容量の切り替えをします.さらにCPUについても**図13**のように切り替えます.それ以外のジャンパ・スイッチは標準状態でかまいません.

次にターゲット・ボードのROMソケットへICEのROMプローブを挿入します.ピン方向を間違えないでください.

そしてCPUの$\overline{RESET}$端子に$\overline{RST}$を,$\overline{M1}$端子に$\overline{M1}$を,SRAMの$\overline{WE}$端子に$\overline{WE}$の各ICクリップを接続します.CPUやメモリがフラット・パッケージなどを使っていて,ICクリップの接続ができないときは,その信号が接続されているコネクタや拡張端子などでかまいません.

次にホスト・パソコンとICEをRS-232-Cのストレート・ケーブルで接続します.ホスト・パソコン側のシリアル端子は,パソコンの機種やディスクトップ型やノート型などの違いでいろいろあるので,ケーブルは各自使うパソコン用のストレート・ケーブルを用意してください.ICE側はD-Sub25ピン・メスです.

そして接続間違いがないかどうかよく確かめてから,ターゲット・ボードの電源を入れます.

● ホスト・パソコン側の準備

まずMS-DOSを起動したら,添付フロッピ・ディスクに入っている環境セットアップ・プログラムSETZVPを実行します.

>SETZVP[リターン]

起動画面を**図14**に示します.ここでは使用するマイコン・ボードのCPUの種類,メモリ・マップ,プログラマブル・ウェイト・ステート・ジェネレータ内蔵のCPUのときは,そのウェイト数などを設定します.

設定が完了したら終了してください.

これ以降,同じマイコン・ボードを使う場合は再設定は必要ありません.メモリ・マップを変えたり,別のマイコン・ボードを使用するときにもう一度設定し

てください．

そしてZVPを起動します．起動方法は，

　>ZVPM[リターン]

とするだけです．オプション・スイッチは特に必要ありません．

● ZVPの画面構成

図15に起動時の画面のようすを示します．

①をダイアログ・ウィンドウと呼び，コマンドの入力はここで行います．②をディスプレイ・ウィンドウと呼び，逆アセンブル・リストが表示されます．③をレジスタ・ウィンドウと呼び，各レジスタの値やフラグのようすを示します．④をダンプ・ウィンドウと呼び，メモリの16進ダンプ表示をします．④'はダンプ・ウィンドウのASCII文字表示画面です．

⑤はブレーク・ウィンドウで，ブレーク・ポイントが表示されます．⑥はZVPの動作モードの表示で，ターゲット・ボードと通信しているときは通信中，Gコマンドでプログラム実行中は実行中，コマンド待ちの状態のときは休止中と表示します．

ZVPの使い方

● プログラムのロード

まず何はともあれ，デバッグしたいプログラムをロードしなければなりません．ZVPでロードできるプログラム・ファイル形式は，インテルHEXファイル形式です．またファイル名が同じで，拡張子が.SYMというシンボル・ファイルがあると，そのファイルも読み込みます．

カーソルがダイアログ・ウィンドウにいるときに，F2を押すとロード・ファイル形式の選択画面になります．ここでカーソルをHEX形式に移動しリターン・キーを押すと，HEXファイルの一覧が表示されます．カーソル・キーで読み込みたいファイルを選択しリターン・キーを押せば読み込みます．

またLコマンドから読み込むときは，ダイアログ・ウィンドウで，

　ZVP>L　ファイル名　ファイル名 [リターン]

としても読み込めます．

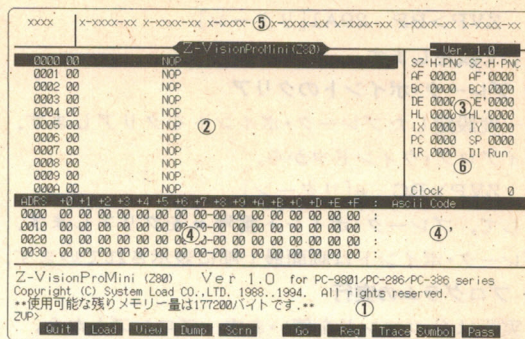
● ダンプ・ウィンドウの表示/編集

ダンプ・ウィンドウに任意のアドレスのデータを表示させたいときは，ダイアログ・ウィンドウでDコマンドを使います．

　ZVP>D　8000[リターン]

これでアドレス8000h以降のメモリが16進ダンプ画面で表示されます．

内容を編集したいときは，ダイアログ・ウィンドウでF4キーを押すとカーソルがダンプ・ウィンドウに移動します．ここで数値を入力すればOKです．また

〈図15〉Z Vision PROminiの起動画面

カーソル・キーで移動すると，表示範囲がスクロールします．またもう一度F4キーを押すとダイアログ・ウィンドウにカーソルが戻ります．

● ディスプレイ・ウィンドウの表示/編集

ディスプレイ・ウィンドウに任意のアドレスのリストを表示させたいときは，ダイアログ・ウィンドウでUコマンドを使います．

　ZVP>U　100[リターン]

これでアドレス0100h以降の命令がアセンブル・リストで表示されます．またダイアログ・ウィンドウでF5キーを押すと，カーソルがディスプレイ・ウィンドウに移動し，カーソル・キーで上下方向に移動するとスクロールします．もう一度F5キーを押すとダイアログ・ウィンドウにカーソルが戻ります．

命令を書き換えたいときは，ダイアログ・ウィンドウから，

　ZVP>A　200[リターン]

とすると，アドレス200h番地から，1パス・アセンブルができます．1パス・アセンブルとは，1行入力するごとに機械語に変換してくれる機能です．何も入力しないでリターン・キーを押すと，ダイアログ・ウィンドウに戻ります．

● レジスタの値の変更

レジスタの値を変更するときは，ダイアログ・ウィンドウからF7キーを押します．するとカーソルがレジスタ・ウィンドウに移動し，カーソル・キーで変更したいレジスタに移動できます．そして数値を入力すると値を変更できます．リターン・キーを押すか，もう一度F7キーを押すとダイアログ・ウィンドウに戻ります．

またダイアログ・ウィンドウから直接，

　ZVP>R　PC　8000[リターン]

と，レジスタ名と値を指定しても設定できます．

● ブレーク・ポイントの設定

ブレーク・ポイントとは，実行を止めたいアドレスのことで，

ダイアログ・ウィンドウから，

ZVP>BP　29A1[リターン]

として設定します．

● **ブレーク・ポイントのクリア**

一度設定したブレーク・ポイントをクリアします．ダイアログ・ウィンドウから，

ZVP>BC　n[リターン]

として，ブレーク・ポイント番号 n をクリアします．ブレーク・ポイントは画面最上部に表示されています．

● **プログラムの実行**

実行はGコマンドを使います．プロンプトから，

ZVP>GO[リターン]

とすると，アドレス0000hから実行を開始します．アドレスがブレーク・ポイントになったら実行をやめてモニタに戻ってきます．

また直接ブレーク・アドレスを指定することもできます．

ZVP>GO　290[リターン]

この例では，アドレス0000hから実行を始めて，アドレスが290hになるとブレークしてモニタに戻ってきます．

● **ステップ実行**

ステップ実行とは，命令を1ステップずつ実行していくものです．1命令の実行が終わるたびに画面の各

〈表8〉Z Vision PROmini のコマンド一覧

コマンド	内　容	書　式
A	1パス・アセンブルを行う	A[〈アドレス〉]
BP	ブレーク・ポイントの設定	BP[〈n〉]〈アドレス〉[〈回数〉]
BD	ブレーク・ポイントの無効化	BD　n_0[n_1[n_2…]]
BE	ブレーク・ポイントの有効化	BE　n_0[n_1[n_2…]]
BC	ブレーク・ポイントの削除	BC　n_0[n_1[n_2…]]
BL	ブレーク・ポイントのリスト	BL
BI	ブレーク・ポイントの通過回数の再設定	BI　n_0[n_1[n_2…]]
D	メモリ・ダンプの表示と書き換え	D[〈モード〉][〈開始アドレス〉[〈終了アドレス〉]]
F	メモリをデータで埋める	F〈開始アドレス〉〈終了アドレス〉d_1[d_2[d_3…]]
G	プログラムの実行	G[〈開始アドレス〉[〈ブレーク・ポイント〉]]
ID	指定ポートからデータを読み込む	ID〈I/O アドレス〉
L	インテルHEXファイルの読み込み	L[〈HEX ファイル〉][〈シンボル・ファイル〉]
MV	メモリのコピー	MV〈開始アドレス〉〈終了アドレス〉〈転送先アドレス〉
OD	指定ポートへのデータの書き込み	OD〈I/O ポート〉〈出力データ〉
P	パス・トレース	P[〈回数〉]
Q	終了	Q
R	レジスタ，フラグの設定	R[〈reg〉][〈値〉]or RF〈flgs〉[〈flgs〉[〈flgs〉……]]
SL	シンボルの表示	SL[〈シンボル〉]
SC	シンボルの削除	SC〈シンボル〉
SS	シンボルの登録	SS〈シンボル〉〈値〉
T	ステップ・トレース	T[〈回数〉]
U	逆アセンブル・リストの表示	U[〈アドレス〉]
WH	インテルHEXファイルの出力	WH[〈ファイル〉[〈開始アドレス〉[〈終了アドレス〉]]]

注）ダイアログ・ウィンドウのコマンドと指定形式．[]パラメータは省略可．
〈〉でくくられたパラメータは0～0FFFFh（16進）の指定が可能．

(a) コマンド一覧

10進数	先頭が%で始まる場合	%124（10進数の124）
16進数	先頭が0～9で始まる場合 A～Fの文字で始まる場合は先頭に0を付ける	8000（10進数の32768） 0FFFFh（10進数の－1）
シンボル	先頭が数字以外の文字（ただし小文字は大文字に変換される） シンボルに使用できる文字 A～Z 0～9 $. ? @ _	BC（"BC"というラベル名）
レジスタ	先頭が#で始まり，以下のレジスタ名文字列がつづいたとき A,B,C,D,E,H,L,I,R,BC,DE,HL,IX,IY,PC,SP A',B',D',E',H',L',BC',DE',HL'	#BC（BCレジスタに格納されている値）
演算子	＋のみ使用可能	8000＋10（16進数で8010h）

(b) 定数，数値の表現

ウィンドウに直後の状態が表示されるので，レジスタの変化のようすが手に取るようにわかります．

まずRコマンドなどで実行開始アドレスをPCレジスタに設定します．そしてダイアログ・ウィンドウでF8キーを押すと，PCレジスタで示されるアドレスの命令を1命令実行して戻ります．各レジスタやメモリ内容は，その実行した命令によって変化します．当然PCレジスタも次の命令のアドレスを示します．

ですから一度押すと次の命令を，さらに押すとまた次の命令をという具合に，1命令ずつ実行を繰り返します．

● **連続ステップ実行**

先ほどのステップ実行と同様にPCレジスタに実行開始アドレスを設定してから，ダイアログ・ウィンドウでSHIFTキーを押しながらF8キーを押します．ESCキーを押すまで連続的にステップ実行します．

● **パス・トレース実行**

ステップでは，サブルーチンがコールされても，サブルーチンも1ステップずつ実行していました．パス・トレースとはサブルーチンなどのコールがあったときは，そのサブルーチン内はブレークせずにリアルタイムで処理するトレースです．

サブルーチン・コールをそれ自体があたかも1命令であるかのようにステップ実行できるので，メイン・ルーチンの実行の流れをチェックしたいときに使用します．これもステップ実行と同様にPCレジスタにアドレスを設定したら，ダイアログ・ウィンドウでF10キーを押します．

● **連続パス・トレースの実行**

パス・トレースを連続で行うモードです．使い方は連続ステップと同様です．ESCキーで終了します．

● **終了**

ダイアログ・ウィンドウで，F1キーまたは，

ZVP＞Q［リターン］

とすると，終了の確認を聞いてくるので，Yを押せば終了します．

表8にZVPmini版のコマンドの一覧を示します．

● **デバッグの手順**

まずデバッグするプログラムをロードします．ROM領域に対しても読み書きできるので，プログラムがアドレス0000hから始まるものでもかまいません．

デバッグのコツはブレーク・ポイントの設定とトレースの使い方にあります．正しく実行することがわかっている部分をトレースしても時間の無駄です．そこでここから先の動作を調べたいというアドレスにブレーク・ポイントを設定し，実行してブレークさせます，そこからステップ実行やトレースをさせて，レジスタやメモリの変化を調べるというやり方が一般的です．

〈藤丘勝信〉

◆参考文献◆

(1) Z Vision PRO ユーザーズマニュアル，㈲システムロード．

頒布ディスクの内容と申し込み方法

● **頒布ディスクについて**

本特集に掲載したリストをディスク頒布します．

ディスクの内容は，第1章から第11章までにでてきたすべてのリストと，Z80用アブソリュート・クロス・アセンブラZAAです．

ZVPについては目次裏の申し込み方法をご覧ください．

● **頒布ディスクの申し込み方法**

下記申し込み用紙に必要事項を記入し，代金同封のうえ**現金書留**で，右記宛先までお送りください．

▶**頒布価格**　3,000円（税，送料込み）

▶**頒布メディア**

3.5インチ2HD（1.25Mフォーマット）

3.5インチ2HD（1.44Mフォーマット）

5インチ2HC（PC9801読み込み可）

▶**申し込み期限**

1998年12月31日（当日消印有効）

▶**申し込み先**

〒170　東京都豊島区巣鴨1-14-2　CQ出版㈱

トランジスタ技術SPECIAL No.49

ディスク頒布係

● **トランジスタ技術　SPECIAL No.49 ソース・リスト申し込み用紙**　TRSP49Z

送り先ご住所：〒

お名前：　　☎（　　）　ー

▶希望メディア（✓印を付けてください）

□3.5インチ2HD(1.25M)

□3.5インチ2HD(1.44M)

□5インチ2HC

●本書掲載記事の利用についてのご注意 ── 本書掲載記事には著作権があり，また工業所有権が確立されている場合があります．したがって，個人で利用される場合以外は所有者の承諾が必要です．
　また，掲載された回路，技術，プログラムを利用して生じたトラブル等については，小社ならびに著作権者は責任を負いかねますのでご了承ください．

●本書に関するご質問について ── 文章，数式等の記述上で不明な点についてのご質問は，必ず往復はがきか返信用封筒を同封した封書にてお願いいたします．ご質問は著者に回送し直接回答していただきますので，多少時間がかかります．また，本書の範囲を超えるご質問には応じられませんので，ご了承ください．

トランジスタ技術 SPECIAL No.49

1995年1月1日 初版発行
1997年7月20日 第3版発行

編集人
発行人 蒲生良治
発行所 CQ出版株式会社 〒170 東京都豊島区巣鴨1-14-2
電話 03-5395-2121(出版部)，03-5395-2141(販売部)
振替 00100-7-10665

(定価は表四に表示してあります)

印刷・製本 三晃印刷株式会社

乱丁，落丁本はお取り替えいたします．

Printed in Japan